SOFTWARE—PRACTICE AND EXPERIENCE

Softw. Pract. Exper. 2011; **00**:1–35

Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

MicroMAIS: Executing and Orchestrating Web Services on Constrained Mobile Devices

C. Cappiello[1], M. Comuzzi[2], B. Pernici[1], P. Plebani[1*] and S. Yadav[3]

[1]*Politecnico di Milano, Piazza Leonardo da Vinci 32 - 20133 Milano, Italy*

[2]*Eindhoven University of Technology, P.O. Box 513 - 5600MB Eindhoven, The Netherlands*

[3]*Texas A&M University, TAMU 3112, College Station, TX 77843-3112, USA*

SUMMARY

Mobile devices with their more and more powerful resources allow the development of Mobile Information Systems, in which services are not only provided by traditional systems, but they are also autonomously executed and controlled in the mobile devices themselves. Services distributed on autonomous mobile devices allow both the development of cooperative applications without a back-end infrastructure and the development of applications blending distributed and centralized services. In this paper, we propose MicroMAIS: an integrated platform for supporting the execution of Web service based applications natively on a mobile device. The MicroMAIS platform is composed of mAS and μ-BPEL. The former allows the execution of a single Web service, whereas the latter permits the orchestration of several Web services according to the WS-BPEL standard. Copyright © 2011 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Mobile Information Systems, WS-BPEL, Micro Application Server

*Correspondence to: Pierluigi Plebani, *Dipartimento di Elettronica e Informazione*, *Politecnico di Milano*, Piazza Leonardo da Vinci, 32 - 20133 Milano, Italy. E-mail: plebani@elet.polimi.it

1. INTRODUCTION

Mobile devices are continuously increasing their capacity and are now able to execute more sophisticated and complex applications than before. Several solutions based on Service Oriented Architectures consider mobile devices as the means to provide pervasive solutions to the users [1]. Nevertheless, mobile devices often hold the role of service consumers instead of service providers. Due to their limitations in terms of available resources (e.g., power, memory), mobile devices run applications able to connect to services installed on traditional devices to exploit their functionalities. In this scenario, the service consumption requires a stable connection with the service provider. In a mobile environment this might not be guaranteed, due to the nomadic nature of mobile devices and different contexts of use.

In this paper, we introduce a Web service based platform, i.e., MicroMAIS [†], developed as a means for supporting Mobile Information Systems [2]. MicroMAIS is composed of two modules, namely mAS (micro Application Server) and μ-BPEL. These modules, designed and developed according to the most common Web service standards, are capable of supporting the execution of simple and composite Web services directly on a mobile device. Using mAS and μ-BPEL, we aim at enabling the execution of Web service based applications natively on mobile devices.

More specifically, mAS is an application server capable of running on the least capable mobile device configuration (CLDC). Java-based Web services can be deployed and executed on mAS. μ-BPEL is a Business Process Execution Language (WS-BPEL) engine designed for supporting service orchestration in a mobile environment. By allowing the execution of both stateless and stateful services and the possibility to manage multiple instances, the MicroMAIS platform allows flexibility of operation in unreliable, nomadic communication environments. Also, while there has been significant previous work in the areas of mobile Web service execution [1, 27, 29, 33, 34] and orchestration [43, 44, 45], most proposals do not rely on the least capable device configuration (CLDC), but they rather consider more powerful mobile devices.

[†]For more information on the MAIS Project (Multichannel Adaptive Information Systems) see http://www.mais-project.it

Prepared using speauth.cls
DOI: 10.1002/spe

The design of mAS and μ-BPEL preserves as much as possible the interoperability with traditional Web service based applications. Specifically, services running on mAS can be invoked as traditional Web services and processes deployed in μ-BPEL can invoke traditional Web services as well. As a consequence, MicroMAIS allows the execution of distributed applications involving different types of devices.

The opportunity to execute standard Web services and Web service compositions natively in a mobile device is considered relevant in several scenarios, such as, for example, risk management and logistics [3, 4].

Risk management in dangerous good transportation, in particular, requires orchestration of processes executed on mobile devices [3]. In this scenario, mobile information systems, such as sensors on the carrier, are able to communicate position and status information via GPRS or UMTS protocols to the Geographical Information Systems (GISs) in the headquarters of the transportation company. GISs collect position and status information and mash it up with other information, such as the presence of water sources, e.g. rivers and basins, or the presence of sites of public interest, e.g. schools or arenas, to determine the best set of rescue activities to be performed on the ground in case an accident occurs.

The process enacting rescue activities coordinates different actors on the field, such as private rescue teams, representatives of the transportation company, and teams from public rescue agencies (e.g. fire brigades), acting in a disrupted environment with very strict time constraints. Rescue activities are orchestrated on the field by a manager, who coordinates many teams with different expertise. In the rescue activities, therefore, mobile information systems can be exploited both as server and client applications. In particular, the mobile devices of actors on the field act as client applications of the manager(s) of the rescue activities.

MicroMAIS aims at providing a platform that can be installed on the mobile devices involved in this kind of scenarios to execute SOAP-based services and orchestrate WS-BPEL processes. In this way, even though the communication with traditional devices is also supported, MicroMAIS enables information exchange directly occur among mobile devices.

Prepared using speauth.cls
DOI: 10.1002/spe

The rest of the paper is structured as follows. In Section 2 and Section 3 we discuss mAS and μ-BPEL, respectively. Section 4 introduces an analysis of the performance of our tools when running a BPEL process. After the Section 5 where related work is discussed, Section 6 concludes the paper and discusses possible future work.

2. MAS: MICRO APPLICATION SERVER

The micro Application Server (mAS) is a J2ME-based application server that supports stateless and stateful Web service execution on cellphones, PDAs, and all least-capable CLDC (Connected Limited Device Configuration) micro (or mobile) devices. While most of the solutions in the field of mobile Web service execution considered mobile devices only in the client-side of a client-service interaction [5], with mAS (similarly to [1, 27, 29, 34]) a mobile device can additionally act as a service provider to other mobile devices or traditional devices acting as service consumers.

Resource-constrained mobile devices introduce many restrictions that make it difficult to implement a complete application server on the offered platform. Especially in case of CLDC Java Virtual Machines (JVMs), the limitations are very strong and, consequently, may affect the functionalities offered by mAS. For instance, hot deployment is not possible since all the classes required during the execution of a service need to be included in the same MIDlet installed on the mobile device before the application starts.

Even though various mobile device architectures are augmenting their performance through technology improvement, the amount of available resources for specific classes of devices, such as cellphones, remains limited. Cellphones use the CLDC JVM platform, which comprises a smaller set of classes than other platforms, e.g., CDC JVMs. Therefore, despite running on a CLDC based configuration, mAS provides a rich set of server-based features, as found only on devices with better configurations/resources. Moreover, mAS supports the execution of both stateless and stateful Web services to support the nomadic nature of mobile devices. mAS provisioning enables proper management of the state of conversation with the client, especially in case of network

Prepared using speauth.cls
DOI: 10.1002/spe

disconnections. Running mAS on constrained mobile devices ensures addressing a larger consumer market, owing to CLDC JVM's popular deployment.

mAS design and development are inspired by existing application servers, in particular the J2EE compliant application servers, and aim at achieving complete compatibility with existing Web service standards and support of Web service interaction based on the Simple Object Access Protocol (SOAP). mAS design involves two main actors: (i) the *application provider* which uses the application server to ensure availability of its software components and (ii) the *application client*, which communicates with server software components, in accordance with the server APIs. The application server is responsible for deployment and execution of applications designed and developed using specific programming and deployment models. In mAS, these models are similar to the ones defined for Enterprise Java Beans (EJBs) and specify how the components must be developed and how they must be described, to make the application server aware of the required resources, the supported functionalities, and the supported communication protocols.

2.1. mAS architecture

mAS implements a container-based application server that communicates with clients according to an Axis-like [6] architecture implementing the *Chain of Responsibility* [7] design pattern. mAS architecture relies on three main elements (see Figure 1):

- *SOAP Request and SOAP Response message*: it represents the requests that clients make towards mAS and also describes how mAS responds to the clients. Depending upon the supported application protocol, mAS provides a *Listener* that receives and marshals incoming messages and, in turn, unmarshals and sends the responses back to the client. After the *Listener* does its processing, any mAS module, regardless of the specific application protocol, can access the message using the *MessageContext* object, i.e., the Java memory-model representation of a message. Note that the CLDC JVM allows connections only through sockets. Opening higher level connections is unfeasible.

Prepared using speauth.cls
DOI: 10.1002/spe

- *Request and Response handlers chain*: it allows a step-by-step analysis of the incoming *MessageContext* or a step-by-step creation of the outgoing *MessageContext*. In each step, a specific *Handler* analyzes a part of the message, depending upon its competence. For instance, the *HTTP handler* interprets protocol parameters, extracts the body, and passes it to the next handler (*SOAP Handler*) that realizes the requested service, the action, and the parameters. An exception is raised if the incoming message does not respect the rules of the communication protocols.
- *MService container*: it receives all the information about the service invocation and manages the required Web service life-cycle. The container is in charge of loading the class corresponding to the requested service. Moreover, in case of stateful services, the container needs to identify the client and to manage the resources.

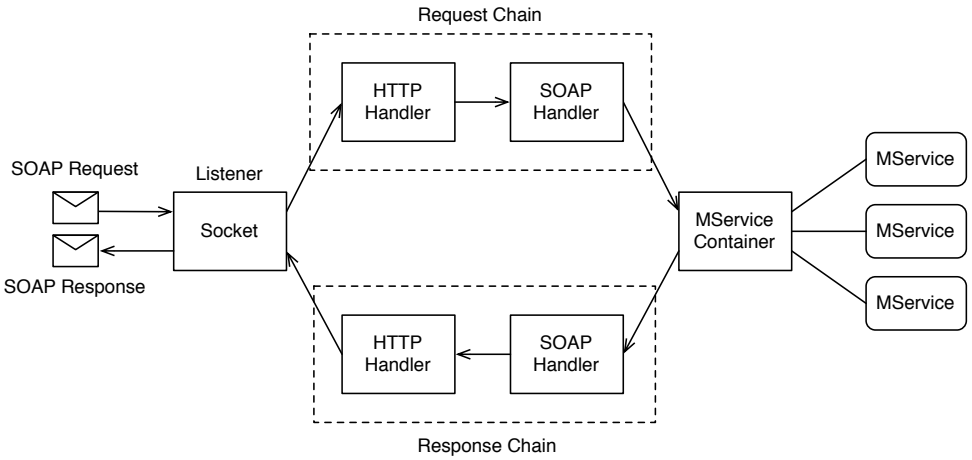

Figure 1. mAS architecture

Although this structure is now usually adopted by application servers, the CLDC JVMs poses new challenges in terms of (i) limited available resources (e.g., memory, computational power) and (ii) limited set of J2ME libraries. Such limitations have a great impact, especially on the design of the MService container that has to allow service deployment, handle the service execution, and manage the connections with the clients. In the next section, we discuss the measures adopted by mAS to overcome these limitations.

Prepared using speauth.cls
DOI: 10.1002/spe

2.2. Data Persistence

Current solutions for data storage in a mobile environment can be grouped into three main classes: *micro-databases*, *reusable solutions*, and *RMS (Record Management Store)-based solutions*.

Micro databases are ad-hoc solutions specifically designed for mobile devices. They usually include GUI-based applications allowing the users to organize and manage small databases. They can support SQL or SQL-like queries and in some cases they also allow ODBC/JDBC connections. An example of micro database is eSQL [8].

Reusable solutions are solutions designed for existing desktop DBMS or superior configurations that can adapt to the constraints of mobile devices. Some cases require a re-design focused on the resources limitations that affect the mobile environments: i.e., lack of memory, less computational power, limited battery life and so on. For instance, SQLite [9] is a DBMS that can be installed and executed on both a traditional PC and a mobile environment. It implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

Finally, *RMS-based solutions* are based on RMS, which provides a mechanism for MIDlets to persistently store and retrieve data [10]. RMS defines a set of *RecordStore*s that, in turn, are a collection of records, which will remain persistent across multiple invocations of the MIDlet. The device platform is responsible for making its best effort to maintain the integrity of the MIDlet's record stores throughout the normal use of the platform, including reboots, battery changes, etc. [11].

Although the first two classes offer a good variety of solutions, all of them work only with CDC JVMs, which, as stated earlier, offer more means than CLDC JVMs. For this reason, data persistence in mAS is based on RMS.

J2ME provides a set of instructions to be able to:

- Manage the RecordStore;
- Monitor the RecordStore;
- Search for data in the RecordStore by filtering and comparing the contents of the records;
- Sequentially access all the records in the RecordStore.

Prepared using speauth.cls
DOI: 10.1002/spe

The contents of the records can only be `Strings` and this requires a marshalling procedure to store complex objects. To simplify the use of RMS, our implementation relies on the Floggy framework [12]. This framework makes use of RMS with usual database management techniques easier. For instance, if a developer would like to store an entire class, instead of performing marshalling and unmarshalling procedures, it delegates these tasks to Floggy by implementing the interface `Persistable` and by exploiting the methods offered by the `PersistableManager` class.

Floggy's main limitation, in the context of the mAS platform, is the inability to handle data structures such as a `HashTable`, which would be useful for storing session based information. In the next section, we detail our approach to overcome this limitation.

2.3. Session management

When executing stateful services, the MService container needs to uniquely identify the sessions as the association between a client and the related service instance. As a consequence, the problem of identification of the client and the definition of links between clients and service instances holds a central role during the service container design. Here, we consider two possibilities: cookies and session ids. These two approaches are similar, since they both assign a unique identification code to every session and put this id in every exchanged message. The difference lies in where each mechanism puts the id in the message: cookies use a proper HTTP attribute and the session is defined in terms of the client id; session ids appear in the HTTP body or, since mAS communicates through SOAP, in the SOAP body. In this case, the same client can activate more than one session.

Although these two solutions are functionally equivalent, in mAS we adopt the solution based on cookies, that is, to put ids in the HTTP header, for two main reasons. First, using a standard HTTP attribute makes the solution compatible with all the HTTP clients as long as they enable the use of cookies; on the contrary, putting the id inside the SOAP body requires to explicitly specify its format and position. Secondly, the id can be retrieved at the very first step inside mAS (i.e., by the

Prepared using speauth.cls
DOI: 10.1002/spe

HTTP listener). This means that we have the opportunity to customize more activities with respect to the connected client: e.g., dispatching the request to the most appropriate chain.

Once a client has been identified, the information about the session is stored in mAS. A session is defined in terms of the client id, the instance of the MService related to the client, the state of the execution, and the session expiration time. Since Floggy does not provide the HashTable data structure, we cannot rely on this class to link the session id to the related information. For this reason, we developed a data structure based on three `Vectors` organized as shown in Figure 2. The first vector, i.e., `memoryKey`, stores the list of active session *ids* and it is used by mAS to browse the active sessions. The same position, in the second vector, i.e., `memoryConnection` is used to index the third vector where the information about the session is finally stored according to the `MasStateKey` class.

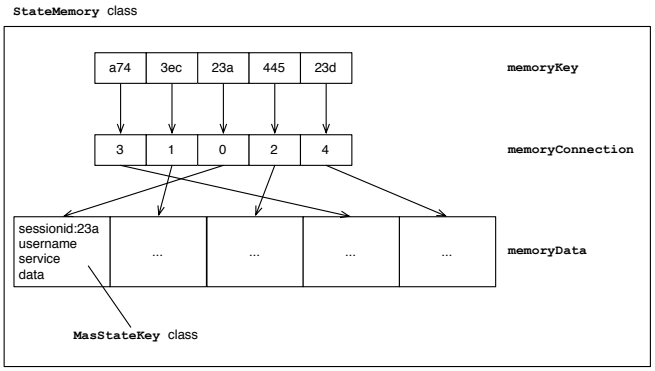

Figure 2. Structure of the `StateMemory` class for session management.

This structure makes the session management more efficient because of three reasons. First, using `Vectors` we can delegate the memory management (when adding and deleting new sessions) directly to a class that exactly knows when to allocate or deallocate objects in memory. Second, we exploit the search mechanisms provided by the `Vector` class, which are already optimized to minimize memory consumption[‡]. Finally, when a session expires, the related object in `memoryData` is no longer considered valid and the cell is available to be overridden with a new

[‡]Indeed, although the session browsing could be done also directly using the `memoryData`, in our solution only one memory access is required instead of two when searching for a session id, since the session id is the element composing the vector and not an object internal to the element composing the vector.

Prepared using speauth.cls
DOI: 10.1002/spe

value; this ensures that in case the same client tries to connect again, either mAS blocks the access due to session expiration, or the client will not find the previous session id in the `memoryKey` vector.

2.4. MService container design

In mAS, a `MService` runs inside a container that manages the entire service life-cycle. We adopt the *Inversion of Control* (IoC) pattern [13], also known as *inversion of dependency*, to identify and to properly instruct the class associated with the `MService` requested by the client.

The IoC is a well known design-pattern adopted to extend frameworks. According to this pattern, all the resources required by a class may not be known at design time. The IoC pattern predicates dynamic binding of these resources at run-time. As such, this pattern is adopted for developing container based applications, such as mAS. A container knows about a component invoked by the user only at run-time. As a consequence, all components available to be called by the container must implement the IoC pattern, implementing a set of pre-defined methods enabling the container to handle their life-cycles. Depending on the adopted IoC style, the container can be aware of the available resources in three different ways:

- *IoC with injection on constructor*: available resources are bound when the object is instantiated;
- *IoC with injection on setters*: the container has a group of setters that are invoked to inform the object of new available resources;
- *IoC with injection on interface*: all resources need to implement a specific interface that makes it possible for the object to communicate with them in a homogenous way.

Several Java based frameworks implementing the IoC with different styles are available, but they all require CDC JVMs. For instance, the Apache Excalibur Fortress [14] project provides a light-weight embeddable IoC container based on the injection on interface style. PicoContainer [15] and its extension NanoContainer implement the IoC based on the injection on constructor style.

Prepared using speauth.cls
DOI: 10.1002/spe

mAS is inspired by these initiatives and assumes that all the deployed services are described by at least two elements: (i) the class implementing the service features and (ii) an XML descriptor that describes those features. As a consequence, all the services that will be executed in the mAS container must comply to both the deployment and programming models described in the following paragraphs. The deployment model is defined by an XML schema, i.e., the descriptor schema, which includes a set of elements describing the main characteristics of the service. The deployment information is used by the container to know how to call and manage the service. The programming model is defined by the interface that all services called in mAS `MService` must implement to be invoked by the container.

2.4.1. Deployment model. The deployment model in mAS follows the same approach used in J2EE, attaching a descriptor file to the compiled class implementing the service. Moreover, all the services must be included in the same jar of mAS, i.e., the MIDlet. This constraint is due to a security policy for J2ME applications that forbids the invocation of classes living outside the jar file of the invoking class. In our case, the mAS container needs to invoke, when requested by the user, one of the deployed services. So, these services must be included in the same MIDlet. Such a security constraint limits the possibility to perform hot deployment that application servers now usually provide.

To elaborate, each deployed service should include a specific folder under the `services` folder. In turn, in each of these Web service specific folders, an `INF` folder includes the XML descriptor, whereas the Java class files of the executable service are stored starting from the service's root.

The service descriptor includes the service logical name, i.e, `display-name`, a simple `description`, the name to be used to invoke the service by the client, i.e., `service-name`, and the `service-type`, which can assume the values *stateless* or *stateful*. Finally, the `service-class` defines the name of the class included in the `service` folder that needs to be loaded by the container, when required.

Prepared using speauth.cls
DOI: 10.1002/spe

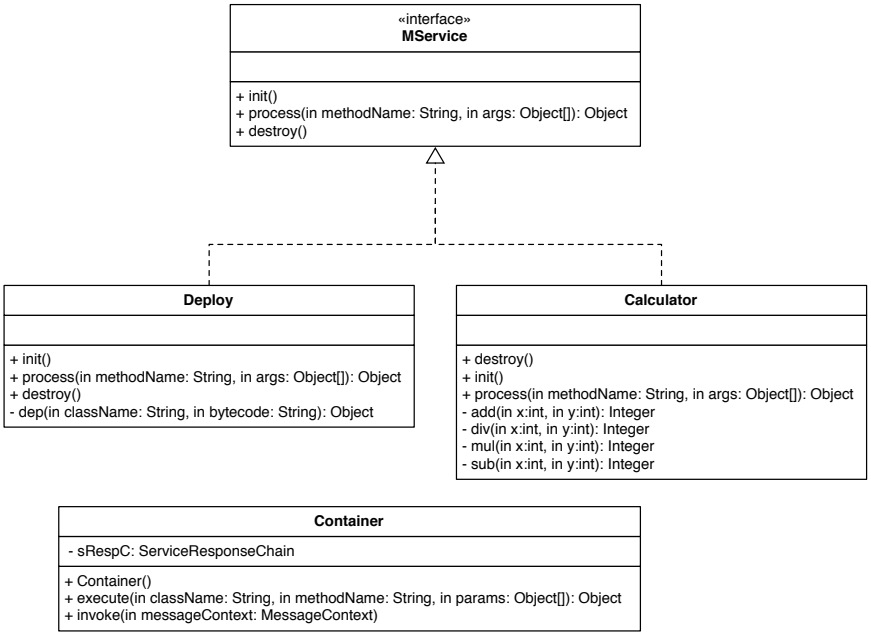

Figure 3. MService interface

2.4.2. Programming model. Mobile Web services in mAS must comply to a specific programming model specified by the `MService` interface. This interface (see Figure 3, referring to an example calculator service) reflects a mobile Web service life-cycle composed by three states: *init*, *process*, *destroy*. As in a servlet life-cycle, *init* and *destroy* allocation and deallocation of resources, respectively, whereas *process* concerns the mobile Web service application logic.

Due to the limitations of CLDC JVMs, the `MService` container is not able to obtain the list of methods provided by a class since the method `Class.getMethods()`, required to accomplish this, is not supported. As a consequence, the mAS programming model includes in the `MService` interface only a standard method, i.e., `process`, that will be invoked by the container when any of the functionalities of a given Web service are required. Thus, the client-side programmer is in charge of invoking the proper code by specifying an appropriate value for the `methodName` input parameter.

Prepared using speauth.cls
DOI: 10.1002/spe

2.5. Client side

From a client's perspective, a Mobile Web service is equal to a traditional Web service. The Web service is described by a Web Services Description Language (WSDL) interface and can be invoked through an application protocol such as SOAP. The WSDL interface lists all the supported operations and also specifies the information about the input and output data of each operation. At this stage, the WSDL file is not automatically generated by mAS and the Web service developer is responsible for creating and including the file in the MIDlet.

3. μ-BPEL: MICRO WS-BPEL ENGINE

Although there exist several proposals enabling workflow execution on mobile devices [43, 44, 45], all previous work use configuration of the mobile devices more powerful than the one we consider in this paper (i.e., CLDC).

With μ-BPEL, we propose a *micro-orchestrator* that allows orchestrating processes described using WS-BPEL on mobile devices. As discussed in [16], the design of workflows with mobile devices needs to consider some peculiar aspects due to their architectural limitations. In this regard, μ-BPEL supports only a subset of the workflow patterns supported by WS-BPEL. In particular, it does not support enhanced WS-BPEL constructs, such as links, correlation sets, and event handlers (see Section 3.2 for an overview of the WS-BPEL dialect implemented by μ-BPEL).

Table I) shows the set of workflow patterns for service composition identified in [17], the ones supported by WS-BPEL, and the subset supported by μ-BPEL. More in detail, within μ-BPEL we do not consider branches with multiple choices (WP6 in [17]) and the related synchronizing merge (WP7), the possibility of having multiple instances of an activity without synchronization (WP12), deferred choice (WP16), and interleaved parallel routing (WP17).

WP6 and WP7 are not supported also by other standards such as BPML [18] or WSCI [19] [17]. Moreover, they can be implemented in WS-BPEL only with control links, the semantic of which was unclear in WS-BPEL 1.1 and consequently revised in WS-BPEL 2.0. In μ-BPEL, we prioritize the

Prepared using speauth.cls
DOI: 10.1002/spe

Table I. Supported Workflow patterns.

Pattern	**μ-BPEL**	**WS-BPEL**
Arbitrary cycles	n	n
Cancel activity	y	y
Cancel case	y	y
Deferred choice (WP16)	n	y
Discriminator	n	n
Exclusive choice	y	y
Implicit termination	y	y
Interleaved parallel routing (WP17)	n	y
Milestone	n	n
Multi-merge	n	n
Multiple choice (WP6)	n	y
Multiple instances without synchronization (WP12)	n	y
Multiple instances with a priori design time knowledge	y	y
Multiple instances with a priori runtime knowledge	n	n
Multiple instances with no a priori knowledge	n	n
Parallel split	y	y
Sequence	y	y
Simple merge	y	y
Synchronization	y	y
Synchronizing merge (WP7)	n	y

constructs that have remained stable in the WS-BPEL evolution and we leave the implementation of WP6 and WP7 to future work.

Concerning WP12, due to the scarcity of resources in the mobile scenario, we opted for supporting multiple instances only in case the number of instances was known a priori at design time.

Concerning WP17, interleaved parallel routing can be captured in WS-BPEL using serializable scopes. However, the semantic of serializable scopes is not clearly defined in WS-BPEL 1.1 ([17], p. 209).

WP16 (deferred choice) can be implemented using the *pick* construct in WS-BPEL. The *pick* construct and, consequently WP16, is not supported in μ-BPEL. We plan to address this limitation as future work. In practice, μ-BPEL does not support dynamic processes in which activities can be selected on the basis of received messages or temporal alarms.

Concerning communication patterns, at this stage μ-BPEL supports the Request/Reply and One-Way patterns. This means that the process execution can start only if requested by the client. As a consequence, all the processes must include at least one `<receive>` element in the definition, whereas, the `<reply>` element is optional.

In summary, stemming from the analysis of limitations discussed above, μ-BPEL is designed to:

Prepared using speauth.cls
DOI: 10.1002/spe

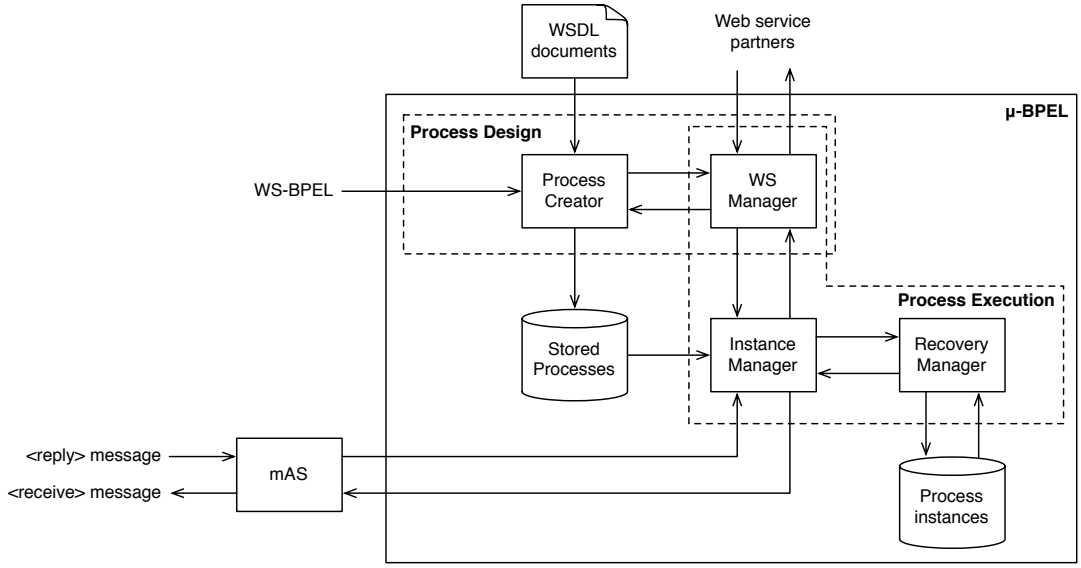

Figure 4. MicroBPEL Architecture.

- support the execution of a Web service-based process;
- manage multiple activities: the orchestrator is able to manage the parallel execution of activities belonging to the same workflow;
- manage multiple instances: multiple process instances can be managed. Instances can refer to the same process or to different processes;
- manage the state of execution: the orchestrator is able to store the execution state (activity state and variable values) of active processes and restore the instances execution after an interruption. Interruptions may occur because of the nomadic nature of the mobile scenario.

3.1. µ-BPEL Architecture

The architecture of μ-BPEL is shown in figure 4. It is possible to identify two overlapping sets of modules: (i) Process Design and (ii) Process Execution. The former set allows the specification of the process to be executed according a specific WS-BPEL document and it comprises the *Process creator* and the *Web Service (WS) manager* modules. The latter set includes the *Instance manager*, the *Recovery manager*, and again the *WS manager* modules. These modules support the execution of the process. The process starts when, according to the WS-BPEL specification, the process client sends a `receive` message having the attribute `createInstance` sets to `yes`.

Prepared using speauth.cls
DOI: 10.1002/spe

In more detail, during the Process Design phase the *Process creator* receives a WS-BPEL specification of the service-based process to execute and translates it in the WS-BPEL dialect used by μ-BPEL (see Section 3.2), in order to build the correct process memory model. If services provided by external partners are needed, the *Process creator* module sends the URL of their WSDL documents to the *WS manager* module. For each service, the *WS manager* identifies the corresponding client stub and sends its specification back to the *Process creator*, which stores the stubs specifications into the *Stored Processes* repository.

The Process Execution phase starts with the activation of the *Instance Manager*, which represents the core of the process engine. The *Instance Manager* retrieves the process model, generates the related process instance, and then starts executing the activities. When the execution of a stub is required, the *Instance Manager* sends a request to the *WS manager* that is responsible to invoke and execute it. The *Instance Manager* module also cooperates with the *Recovery Manager* module that stores and manages all the information about the process state into the *Process instance repository*. This repository contains valuable information especially in case of network disconnection.

3.1.1. Process Design phase. The Process design phase starts with the preparation of the WS-BPEL document that describes the service-based process to execute. μ-BPEL requires that this document is sent to the *Process creator*, which then prepares the process memory model by instantiating all the classes that correspond to the elements composing the process. To this aim, according to the WS-BPEL dialect used in μ-BPEL, the process memory model is revised to eliminate all the classes and attributes no longer supported and to compact the existing ones. Figure 5 shows the related class diagram (some methods details are omitted to improve readability). At run-time, the process memory model provides to the *Instance manager* all the information about the activities to be executed, the variable to be stored, and all the information about the control flow.

In case the process requires the invocation of services offered by external providers, the *Process Creator* sends the WSDL document containing the service description to the *WS Manager*. The *Process Creator* also needs to check if the proper stub is already stored. If so, the process creation

Prepared using speauth.cls
DOI: 10.1002/spe

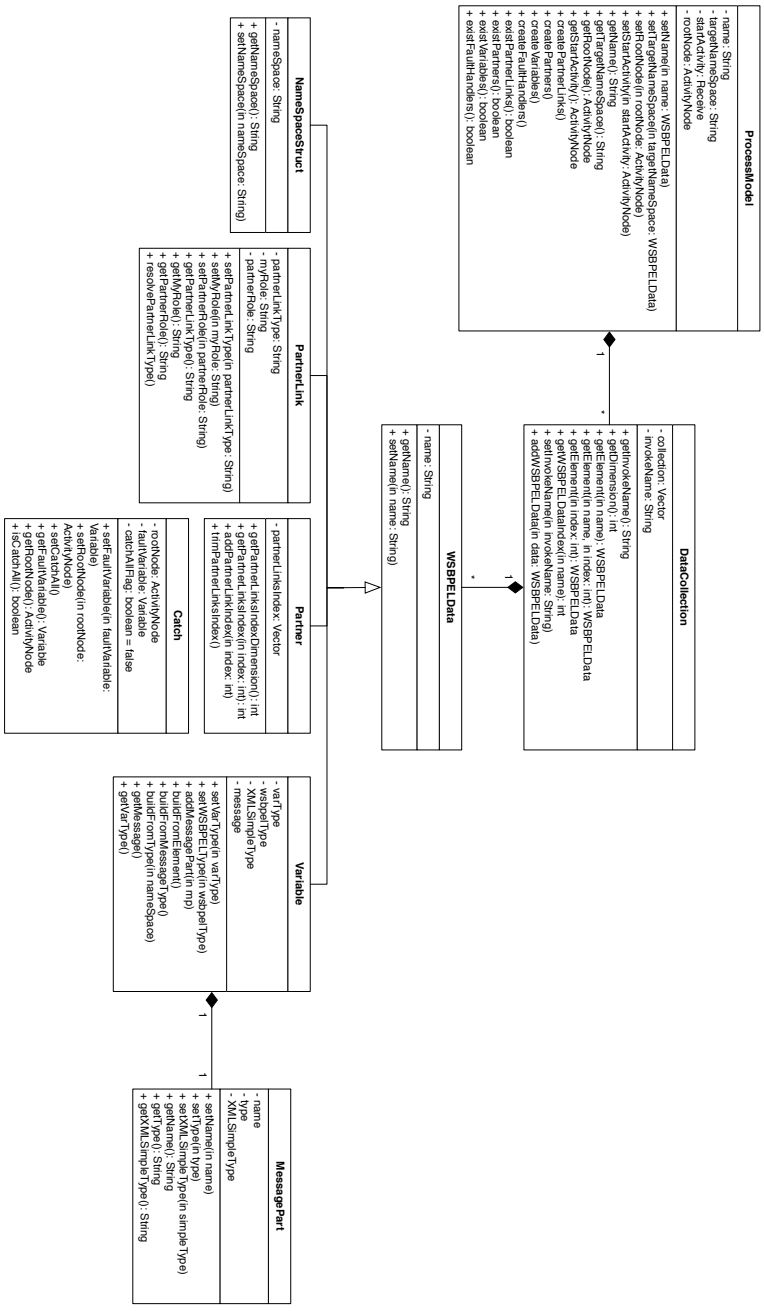

Figure 5. Process memory model for μ-BPEL WS-BPEL dialect.

can continue, otherwise an exception is raised. In such a case, by using external tools usually included in Interactive Development Environment (IDE) for mobile devices, the proper stub is generated and stored by invoking the *Web Service manager*. In the current stage, μ-BPEL assumes that all the external providers are known and the related stubs are already accessible by the platform. Once the memory model is instantiated, it is stored in the *Process model repository*.

Prepared using speauth.cls
DOI: 10.1002/spe

3.1.2. Process Execution phase. The Process Execution phase starts when a user requests to run the process previously created. This request can be sent using either μ-BPEL's graphical interface or its application programming interface (API). The graphical interface allows users to start the process filling a form that is auto generated on the basis of the variables declared in the receive activity that starts the process. When the process terminates, the response message reporting the output variables is directly shown on the device screen to the user. Alternatively, since a process specified in WS-BPEL can be viewed as a Web service, it is possible to create an API that corresponds to the WSDL associated with the whole process. The API exposes only the interactions with the client and uses the functionality discussed in Section 2 to communicate with Web services in mAS.

Regardless of the selected interface, the process is executed by the *Instance Manager* module that is supported by the *WS Manager* module when an invocation of an external Web service is needed. The *Instance Manager* module starts the process execution as a standalone thread to allow the concurrent execution of multiple process instances. When the *Instance Manager* module receives the execution request, it retrieves the right process model and creates a process instance. The first receive activity is then executed by initializing the corresponding variables contained in the instantiation request. The user receives a notification of the beginning of the execution. Each type of activity matches with a specific thread, created from the relative activity class, which is in charge of executing the corresponding tasks. In particular, for structured activities, it is necessary to select which one of the sub-activities has to be executed and to synchronize the activities. In the μ-BPEL architecture the synchronization is performed by a semaphore mechanism. When a structured activity (the parent thread) must be executed, the corresponding thread has to execute the sub-activities (the child threads) in the correct way and in the correct order. The parent thread sets the semaphore variable to the number of child threads that must be immediately executed and then falls asleep. When a child thread finishes executing, it decreases the semaphore value. When the semaphore value is equal to zero, the parent thread awakes and decides what to do. To guarantee the immediate termination of all the activities when an exception is raised or after the execution of a terminate activity, μ-BPEL provides an abort mechanism. There is an abort flag that is set by the

DOI: 10.1002/spe

thread throwing the exception. Each parent thread checks the value of this flag at the time of the wakening. If the value is true, the parent thread will interrupt its own execution and will propagate the alert to the child threads.

3.1.3. Process recovery. The data about variables and the state of a process instance are written in a log by the *Recovery Manager* module. This allows μ-BPEL to recover a process instance abnormally interrupted, because of disconnection from the network or battery failure.

Since the *Recovery Manager* logs also all the possible errors, in case of interruption it is possible to recover the process by executing the remaining part. The log is implemented through the RMS as in mAS so that the persistence of these data is ensured by the mechanisms implemented by the MIDP. There are three types of RMS-based entities in μ-BPEL:

- *RMS for processes*: it is a list of all the process instances started, but not yet terminated. In order to distinguish different instances of the same process, the name of each instance is stored together with a timestamp. When a process terminates (correctly or for an unmanageable error), the corresponding record is deleted. All the entries (not deleted) are used to create a list of the recoverable processes;
- *RMS for activities*: each process instance in execution has a RMS to store the state of its activities. The possible states are RUNNING, TERMINATED and ERROR. When the process terminates, the associated entries are deleted;
- *RMS for variables*: each process instance in execution has a RMS to store the value of its variables. When a process starts, the list of all the variables without an associated value is stored. Once a variable is modified, its value is updated. When a process instance terminates, the associated entries are deleted.

When users want to recover a process, they select one service from the list of the recoverable processes. A new instance of the process is created and launched by the *Instance Manager*, which is aware of being in recovery mode. The variables are created with the values retrieved from the repository or with a null value if the variable is not associated with any values. At this point, the

Prepared using speauth.cls
DOI: 10.1002/spe

Instance Manager starts executing the process. For each activity, it checks the state of the activity in the repository. If the state is TERMINATED or ERROR, the corresponding thread does nothing. If the state is RUNNING or if there is no entry for the activity the thread is normally executed. In case of abnormal termination during a recovery activity, it will be possible to do the recovery another time. The name of the process would not change because the timestamp remains the same as in the first execution.

3.2. WS-BPEL dialect

As previously introduced, μ-BPEL is able to support a subset of the constructs appearing in the WS-BPEL 2.0 specification [20]. For instance, all the elements about complex activities and the modeling of control links are ignored. Also, queries, i.e., XPath expressions, are not supported;

In more detail, Figure 6 lists all the modifications to the WS-BPEL language implemented in the μ-BPEL dialect. Figure 6 contains three different tables that describe (i) the *Unchanged Constructs* that are considered in the same way as in the original WS-BPEL specification; (ii) the *Unmanageable Constructs* that are all the constructs that have been eliminated in the dialect; (iii) the *Modified Constructs* that are all the constructs that have been modified.

4. EVALUATION

Running prototypes of mAS and μ-BPEL are now available and freely downloadable from Sourceforge: `http://masproject.cvs.sourceforge.net/` and `http://microbpel.cvs.sourceforge.net/`, namely. These implementations rely on kSOAP [21] as a SOAP pull-parser for mobile devices, developed by Object Web. This parser is based on kXML [22]: an XML pull parser specifically designed for constrained environments that is suitable for SOAP messages. Indeed, an XML pull parser [23] can read through large XML files efficiently using a small amount of memory.

Prepared using speauth.cls
DOI: 10.1002/spe

Figure 6. Modifications to the WS−BPEL language.

Unchanged Constructs
$< empty >$
$< faultHandlers >$
$< partners >$
$< partnerLinks >$
$< sequence >$
$< switch >$
$< terminate >$
$< throw >$
$< while >$

Unmanageable Constructs
$< compensate >$
$< CompensationHandlers >$
$< correlationSet >$
$< EventHandlers >$
$< link >$
$< pick >$
$< scope >$

Modified Construct	**Comments**
standard-attributes	Deletion of all attributes except for name, that is mandatory
$< process >$	Deletion of the attributes: querylanguage, expressionLanguage, suppressJoinFailure, abstractProcess
$< variables >$	Deletion of the variables of Element type
$< receive >$	Deletion of the constructs $< correlationSets >$
$< reply >$	Deletion of the constructs $< correlationSets >$
$< invoke >$	Deletion of the constructs $< correlationSets >$ and $< compensationHandlers >$
$< assign >$	Deletion of the constructs $< frompartnerLink = "ncname" endpointReference = "myRole\|partnerRole" >< fromvariable = "ncname" property = "qname" >< topartnerLink = "ncname" >< tovariable = "ncname" property = "qname" >$
$< wait >$	Deletion of deadlines
$< flow >$	Deletion of $< link >$

In this evaluation we focus on the response time and the memory occupation during the service execution. In particular, we measured the trend of these parameters with respect to the number of clients and the nature of the executing service (i.e., stateless or stateful). About the response time analysis, to minimize the impact on the total response time given by the execution of the service business logic, we implemented a simple service called `AddService`. This service can run both in stateless and stateful mode, and starting from 0, adds the number received as input. When running in stateless mode, the service will always return the number sent as input. On the contrary, when running in stateful mode, the service will keep incrementing the input values.

The `AddService` is firstly deployed on mAS as a stateless service (by properly setting the service descriptor, see Sect. 2.4). To give the opportunity to measure on-the-fly the memory occupation, we decide to run mAS on the Sun Java Wireless Toolkit 2.5.2 for CLDC emulation environment with 2MByte total memory available.

At client-side, we implemented a thread-based Java application able to open parallel connections to `AddService`. Due to the limited number of possibile simultaneous connections for mAS, we

Prepared using speauth.cls DOI: 10.1002/spe

have 10 concurrent invocations at most. This client had been executed varying the number of parallel connection, from 1 to 10. For each case, in order to have reliable results, we repeated the execution for 10 times.

Figure 7 shows the results of these experiments. The analysis takes into account the execution of Web services running in both stateless and stateful mode varying the number of connected clients. For each case, the figure shows the average response time considering all the connected clients. Focusing on the stateless mode, the average response time varies between $250ms$ and $560ms$ and the trend is linear. It is worth noting that with low number of connected clients the response time is lower than when a single invocation occurs (with 3 connected clients, the response time is the lowest). This trend reflects the computation required by mAS to prepare the system and, in particular, by the container to instantiate the service classes when the first call occurs. As we are considering the stateless mode, the following calls do not require this initial step and they have a lower response time. For higher number of clients, the response time tends to increase and, in some case, the gap is relevant. As discussed in the following, analyzing memory monitor, this situation is due to the execution of the garbage collector that negatively influences the response time.

Switching to the analysis of the stateful mode, the trend of the average response time is exponential and it varies between $411ms$ and $1,65s$. Even in this case, with low number of connected clients, the response time tends to decrease due to the effort required by mAS to organize the internal classes after the first call. Unlike the stateless mode, the instantiation of the service by the container is required for each connected clients and the `StateMemory` structure has to be managed, too. This additional effort results in the increase of the response time that in some cases is more relevant due to the execution of the garbage collector.

Focusing on the memory occupation for mAS, Figure 8 shows the trend on both stateless and stateful mode. It is worth noting that, as the execution of the service requires a very short time, to make the analysis possible the client sends a request every $1s$ and, the service waits for $10s$ after performing the addition. As shown in Figure 8, in stateful mode the occupied memory is generally more than on stateless mode. Moreover, in stateful mode, the garbage collector needed to run two

Prepared using speauth.cls
DOI: 10.1002/spe

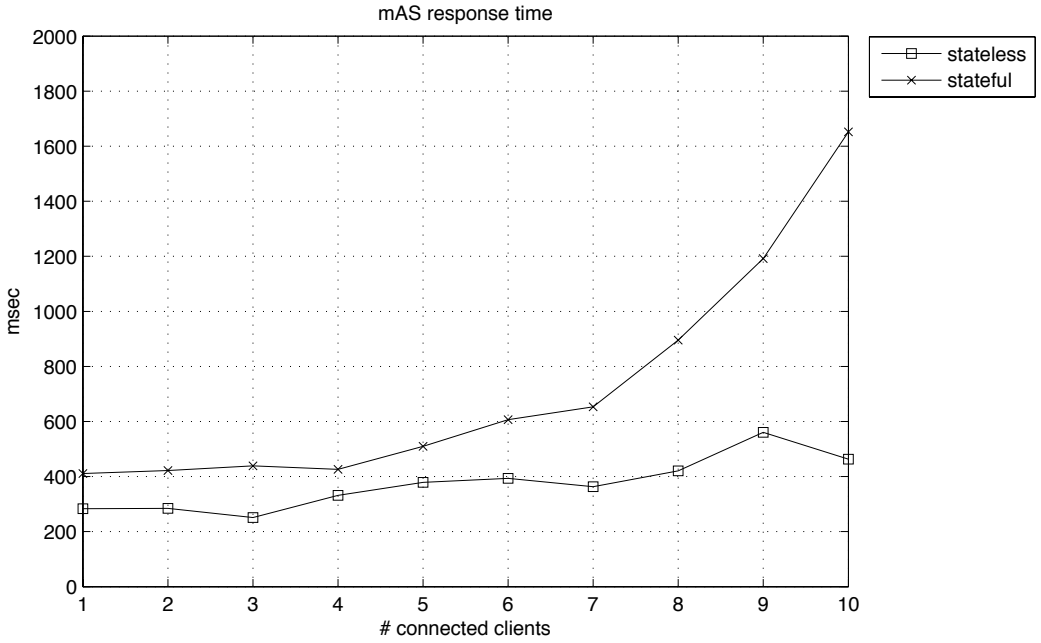

Figure 7. mAS response time varying the number of connected clients.

times. When the first response is sent, since the garbage collector ran, the memory is occupied only by the objects that are really alive and needed to process the requests. More precisely, at $24s$ mAS occupies $188Kbyte$ on stateless mode and $206Kbyte$ on stateful model.

About the evaluation of μ-BPEL, we calculated the amount of time required to perform the most relevant activities in a WS-BPEL process: i.e., receive, reply, assign, and invoke. We decided to concentrate on these activities as the remaining constructs constituting a WS-BPEL process does not affect the overall execution time. For this reason, we performed our tests using the process shown in Figure 9. This process is deployed into the μ-BPEL environment running on a Sun Wireless Toolkit emulator, and executed ten times. On another instance of the emulator, mAS is running and hosts the services invoked by the μ-BPEL process. In this way, we are able to measure the response time and the memory occupation for μ-BPEL only. With this configuration, after each execution of the process, we considered the response time for the relevant activities and the overall response time. As at mAS side, we are able to get information about the time required to execute the invoked service, we excluded this value from the overall response time and the invoke response time. The average of the response times obtained by the tests are shown in Table II for each of the activities.

Prepared using speauth.cls
DOI: 10.1002/spe

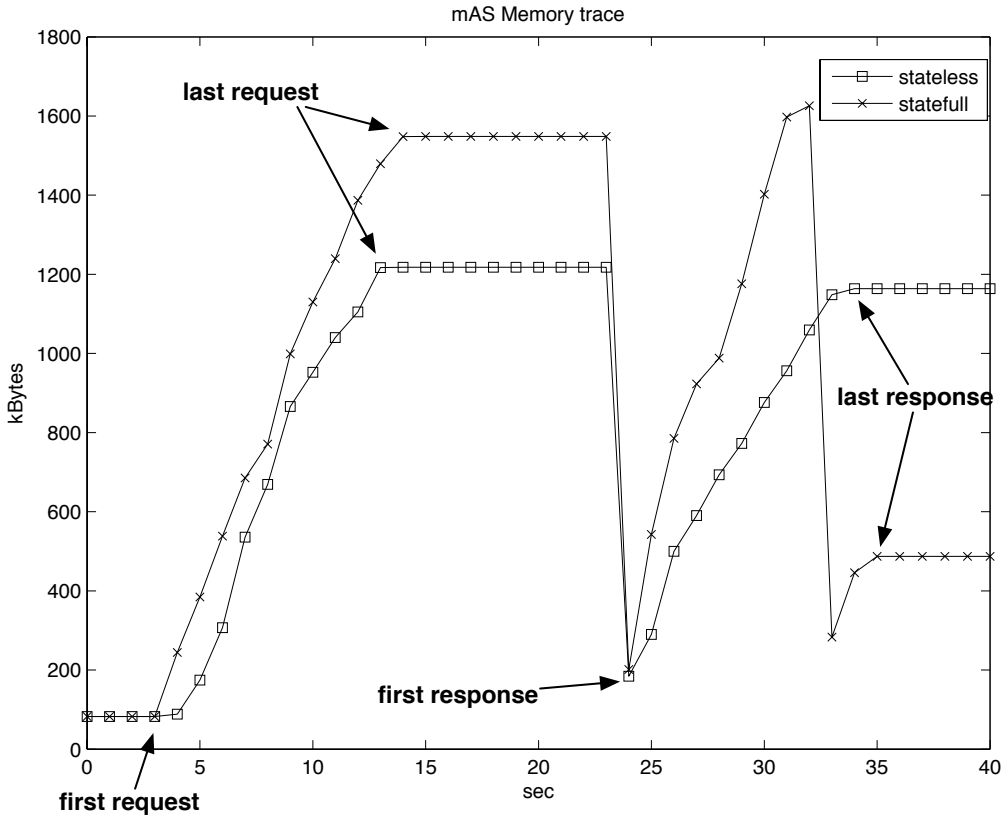

Figure 8. mAS memory occupation with 10 connected clients.

Table II. Response time for main WS-BPEL activities.

Activity	Execution time
Receive	43 ms
Assign	123 ms
Invoke	156 ms
Reply	212 ms

Focusing on the overall response time, the process required 1230 ms on average [§]. The overhead required to prepare the execution of the process to control the execution itself is 573 ms obtained as follows:

$$overhead = 1130\ ms - (43 + 123 \cdot 2 + 156 + 212)\ ms = 473\ ms \quad (1)$$

[§]This value refers to the time elapsed between the receive execution and the reply execution. Thus, the time required select a process and enter input data are not considered

Prepared using speauth.cls
DOI: 10.1002/spe

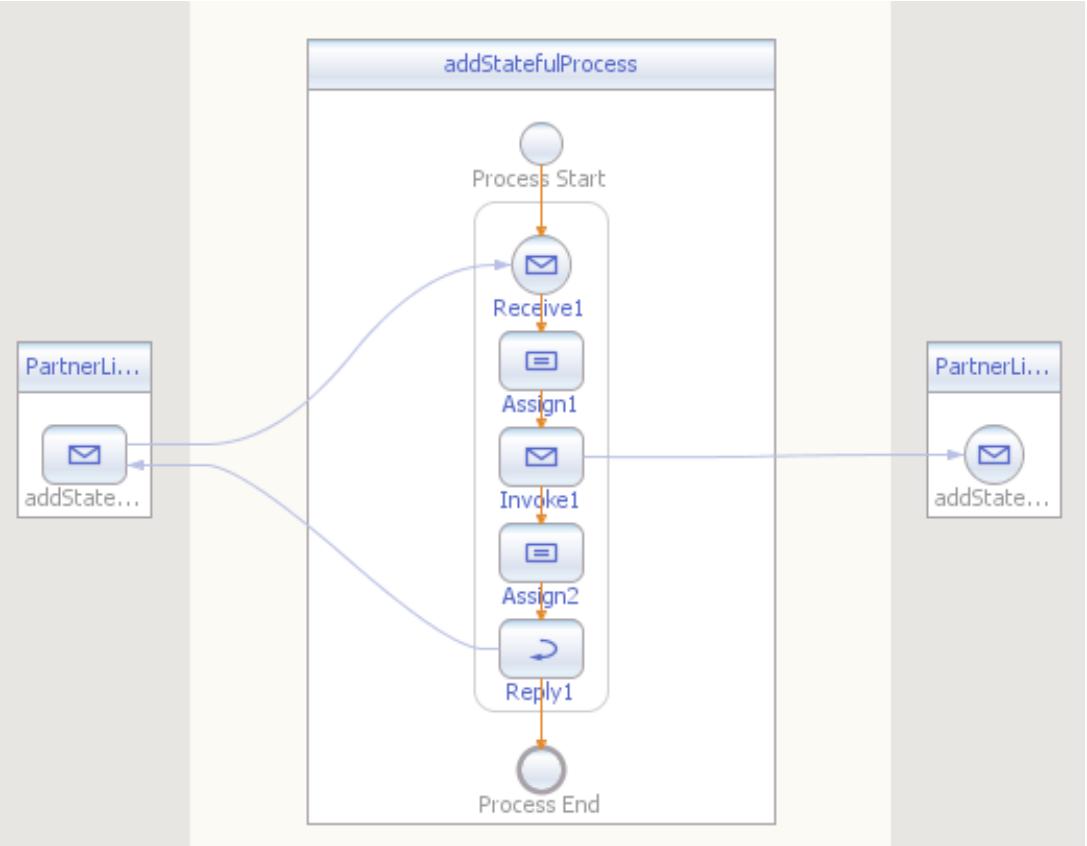

Figure 9. WS-BPEL process used to evaluate μ-BPEL

It is worth noting that about the same overhead time is also observed when executing a variant of the process where the external service invocation occurs five times. In that case, the overall process takes, on the average, $1766ms$ so that the overhead is $485\ ms$ (i.e., $1766 - (43 + 123 \cdot 2 + 156 \cdot 5 + 212) = 485$.

Finally, Figure 10 shows how the memory occupation varies during the process instantiation and execution. In this case, we tested μ-BPEL with the process in Figure 9 and with two variant of it: in the *parallel* variant five parallel 'invoke' are executed, whereas in the *sequence* variant those five invocations occur sequertially.

As shown, the memory occupation is always under $700Kbytes$ and at the end of the process execution is about $170Kbyte$. Even if it is not reported in the figure, also in case of parallel execution, the amount of the memory after the execution remains on that value, but an explicit invocation of the garbage collected is required.

Prepared using speauth.cls
DOI: 10.1002/spe

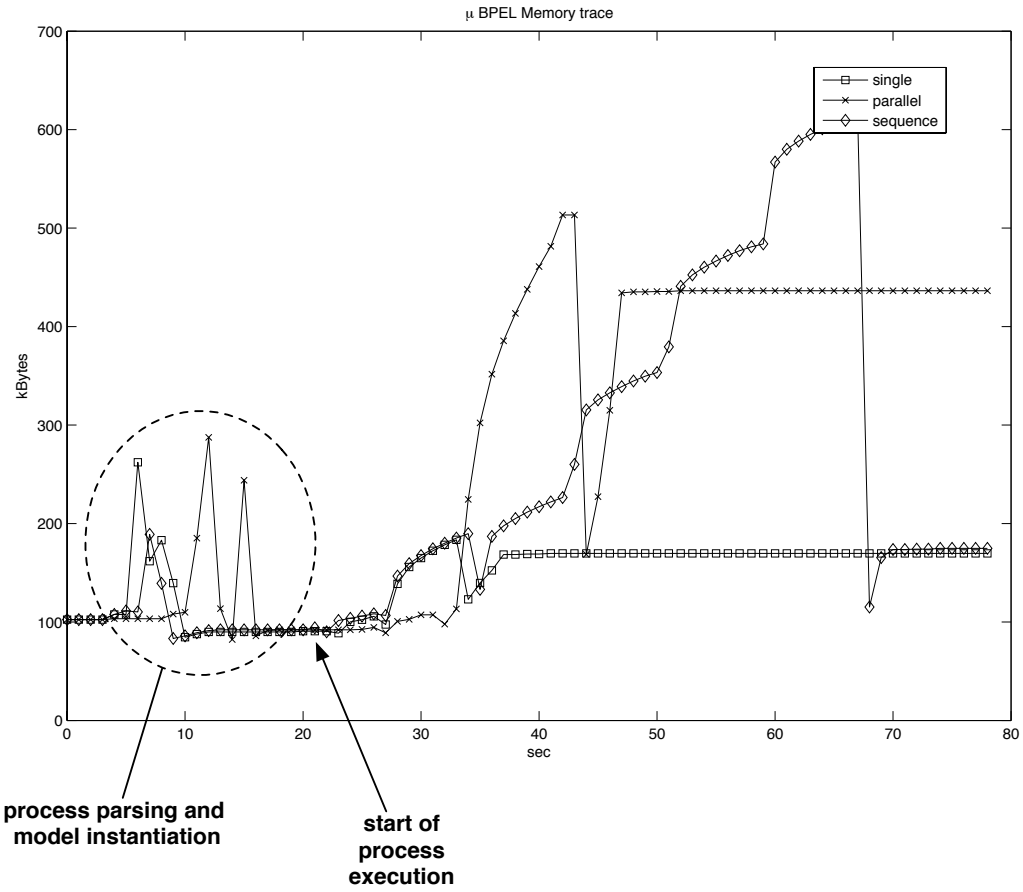

Figure 10. μ-BPEL memory trace.

5. RELATED WORK

As stated in [5], "Mobile Web services are mainly designed so embedded devices can consume the service provided by the server; in other words, mobile Web services are designed from the perspective of the Web services consumer, to enable lightweight devices to share the computing capability and database with the server". This kind of approach has influenced the research on the mobile computing. As a consequence, most of the solutions made available so far by pervasive, ubiquitous, and mobile computing community consider the mobile devices only at the client side of the application. Among the first relevant contributions in this field, we mention [24] and [25], which provide descriptions of the methods and components needed to put together mobile networks that can wirelessly deliver Web content. Nowadays, most of the platforms provide support for invoking

Prepared using speauth.cls
DOI: 10.1002/spe

Web services deployed on traditional servers from mobile devices. For instance, Java only provides the WSA 1.0 (JSR 172) specification that defines an optional package providing standard access from J2ME devices to web services.

Our paper addresses the problem of orchestrating and executing Web services on a mobile device. Although these issues have been scarcely addressed in the literature [26], in the remainder of this section we compare our work with previous contributions addressing the *execution*, *messaging*, and *orchestration* of mobile Web services. Eventually, we also briefly discuss current proposals for mobile Web services execution available in industry.

5.1. Execution of mobile Web services

As regards service execution, Berger et al. in [1] discuss their experience about running Web services on a mobile device. According to them, there are no fundamental impediments preventing the usage of Web services technology on mobile devices. Problems, however, may concern the infrastructures and the mobile device resources. As a confirmation, the authors propose an architecture supporting the publication and invocation of Web services on mobile devices and they demonstrate its real effectiveness. In [27], Gehlen and Pham introduce the implementation of an environment executing Web services on mobile devices. Although the architecture is similar to mAS, it runs on CDC configuration. Moreover, the source code or a downloadable version are not available.

An architecture similar to ours is also proposed in [29], where the authors deploy a Web service handling infrastructure on a SonyEricsson P800 smartphone. The authors choose to deploy their application on PersonalJava platform rather than J2ME CLDC/MIDP, owing to the former's richer application set. However, our platform achieves the same capability with better response times using the constrained J2ME CLDC/MIDP platform. Zaplata et al. in [30] propose an architecture for dynamic adaptation of Web service request based on the resource availability between mobile service provider and consumer. The adaptation may occur through protocol selection, encoding format or other means. The approach in [30] has not been evaluated on advanced platforms. Wolff et. al. [31] highlight the deployment of Web services utilizing the OSGi (Open Source Gateway

Prepared using speauth.cls
DOI: 10.1002/spe

Initiative) for core management. Kim and Lee in [32] propose the use of Web service migration as a means to make Web service availability robust to network or communication faults. The migration policy is driven by the resource availability with respect to target devices capable of hosting Web services. Innovative service advertisement and discovery protocols are proposed in [33], where the proposed architecture is deployed only on Java CDC platform. Eventually, Schmidt et al. [34] present an architecture very similar to ours. Our architecture, however, additionally offers Web service orchestration, incorporating the WS-BPEL industry standard.

5.2. Mobile Web service messaging

Focusing on the messaging sub-system, Park et al. [35] propose an enhanced SOAP message processing specifically designed for mobile Web services. Even if the authors assume that mobile devices still hold the role of the Web service client, the proposed approach performs better than a SOAP-based message communication, which is often the bottleneck of Web service based applications. Similarly, Lai et al. [36] propose a SOAP binding based on UDP to obtain a messaging system 10 times faster than using TCP as a transport protocol.

Deployment of Web services on C++ and Java based platforms have been compared in [37]. In their study, the authors conclude that C++ Web service deployment provides quicker service times because of faster processing of SOAP messages via the gSOAP [46] library. The work in [37], however, exploits the SOAP-based client-server architecture for deployment of their prototype.

Recently proposed alternatives, such as [39], advocate providing REST (Representational State Transfer) based Web services on mobile devices. This claim is motivated by the need for faster processing times lower memory consumption and lower amount of resources available for devices hosting such services. However, REST offers limited flexibility owing to its dependance on HTTP protocol. Hassan et al. in [40] defers heavy computational tasks involved while executing Web services to resourceful workstations. The evaluation in [40] demonstrates an improvement of a factor 2 in response times. However, the authors do not specify the deployment platform. Additionally, our focus is to dispense with workstations altogether making Web services accessible

Prepared using speauth.cls
DOI: 10.1002/spe

anywhere. Glombitza et. al. [41] outline a new protocol called LTP (Lean Transport Protocol), for device-oblivious communication requiring consumption of Web services. LTP employs message compression without overloading device resources.

5.3. Orchestration of mobile Web services

When considering the execution of a complex application based on the orchestration of several Web services, the available solutions are even less. Vimoware [43] is a flexible toolkit for developers to develop Web services and to define and execute collaborative processes with customized scenario. This work can be compared to ours, since μ-BPEL involves a lightweight flow language and allows mobile devices to exchange messages for collaboration. However, Vimoware allows the mobile devices to execute Web service, but a mobile device cannot manage the orchestration as in μ-BPEL. To the best of our knowledge, Sliver [44] and ROME4EU [45] are the only WS-BPEL orchestrators running on mobile devices. With respect to μ-BPEL, Sliver supports more workflow patterns (13 instead of 9), but it runs on a CDC configuration that has less limitation than CLDC on which μ-BPEL can run. ROME4EU is able to support all the BPEL 2.0 specification, thus it can support the 14 workflow patterns that also WS-BPEL supports (see Table I). Moreover, Rome4EU is based on Microsoft .NET Compact Framework that can run only on Windows based mobile platforms, whereas μ-BPEL is based on a CLDC profile that is supported by all the mobile platforms.

5.4. Industry standards and proposals

The OMA (Open Mobile Alliance) group [28] aims at delivering technical specifications that enable interoperable solutions when Web services are invoked from or provided by a mobile device. In particular, while kSOAP2 is a mature library for SOAP based Web service clients running on mobile devices, we did not find a similarly mature proposal for enabling the provisioning of Web services on mobile devices.

DPWS (Devices Profile for Web Services) [42] is an industry standard architecture for specifying hosted and hosting services on resource constrained devices. Currently, Windows Vista and Windows Embedded CE6R2 platform provide support for DPWS as a component of WSDAPI.

Prepared using speauth.cls
DOI: 10.1002/spe

uDPWS [47] is an extension of DPWS tailored for memory-efficient networked embedded systems and wireless sensor networks.

The work in [38] compares the performance of Web service deployment on three major operating systems currently present in the market viz. Symbian OS, Android, and Windows Mobile. Various platforms (such as J2ME CLDC/MIDP) may be ported over either of these operating systems for an improved performance. The study measures the tradeoff between performance and scalability of the mobile operating systems.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced a mobile platform for supporting the execution of Web service based applications that includes both a single Web service and a composition of Web services. The two modules composing the MicroMAIS platform, mAS and μ-BPEL, have been developed by considering all the common Web service standards in order to ensure the interoperability with existing platforms.

The main rationale behind mAS and μ-BPEL is to shift the role of a mobile device from being only the service consumer to being also a service provider. Due to the nature of mobile devices, the limitation about memory, virtual machines, and connectivity deeply affects the architecture and the functionalities of MicroMAIS. Nevertheless, we propose a complete framework that is able to both execute standard Web services and composite Web services orchestrated according to a WS-BPEL process. All of these functionalities are provided on a J2ME virtual machine that provides a small subset of classes with respect to the JVM usually installed on traditional devices. Moreover, we decided to rely on a CLDC JVM (instead of a CDC JVM) to be sure that our platform can be executed on any kind of mobile device on which a Java Virtual Machine can be installed.

To improve the current release of the two modules, we would like to extend the mAS functionalities with a management framework allowing the remote deployment of a Mobile Web service and automatically creating a WSDL file. Finally, we will investigate the porting of MicroMAIS to more recent platforms, such as Android.

Prepared using speauth.cls
DOI: 10.1002/spe

ACKNOWLEDGEMENTS

The authors would like to thank all the students that helped them in developing mAS and μ-BPEL. A special thank to Matteo Galli, Giovanni Furnari, Tommaso Codella, Paola Sandrinelli, Matteo Sansalone. This work has been partially founded by the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

Prepared using speauth.cls
DOI: 10.1002/spe

REFERENCES

1. Berger S, McFaddin S, Narayanaswami C, Raghunath M. Web Services on Mobile Devices - Implementation and Experience. IEEE Computer Society: Los Alamitos, CA, USA, 2003; 100.
2. Pernici B ((ed.)). *Mobile Information Systems: Infrastructure and Design for Adaptivity and Flexibility*. Springer-Verlag New York, Inc.: Secaucus, NJ, USA, 2006.
3. Baresi L, Braga D, Comuzzi M, Pacifici F, Plebani P. A service-based infrastructure for advanced logistics. *Proceedings of the 2nd International Workshop on Service Oriented Software Engineering: in conjunction with the 6th ESEC/FSE joint meeting, IW-SOSWE 2007, Dubrovnik, Croatia*, 2007; 47–53.
4. Brioschi M, Cappiello C, Legnani M, Maurino A, Mussi E, Pernici B, Simeoni N. The MAIS prototype for the risk evaluation of archaelogical heritage. Demo presented in the Developers Track of the 15th International World Wide Web Conference, Edinburgh 2006.
5. Shu Fang R. Designing mobile Web services. http://www.ibm.com/developerworks/wireless/library/wi-websvc/ 2006.
6. Apache Software Foundation. Apache Axis 2 User's Guide. http://ws.apache.org/axis2/1_5_1/userguide.html 2009.
7. Gamma E, Helm R, Johnson RE, Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley: Reading, MA, 1995.
8. Vieka Technology Inc. eSQL Database Engine. http://vieka.com/esql.htm.
9. SQLite. SQLite Documentation. http://www.sqlite.org/docs.html.
10. Gosh S. J2ME Record Management Store. http://www.ibm.com/developerworks/library/wi-rms/ May 2002.
11. Java Community Process. JSR-000118 Mobile Information Device Profile 2.0. http://jcp.org/aboutJava/communityprocess/final/jsr118/ November 2002.
12. Floggy Open Source Group. Floggy. http://floggy.sourceforge.net.
13. Johnson RE, Foote B. Designing Reusable Classes. *Journal of Object-Oriented Programming* June/July 1988; **1**(2):22–35.
14. Apache Software Foundation. Excalibur. http://excalibur.apache.org/.
15. PicoContainer Committers. Picocontainer. http://www.picocontainer.org/.
16. Jing J, Huff K, Sinha H, Hurwitz B, Robinson B. Workflow and Application Adaptations in Mobile Environments. *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, WMCSA '99, IEEE Computer Society: Washington, DC, USA, 1999; 62–69.
17. Wohed P, Aalst W, Dumas M, Hofstede A. Analysis of Web Services Composition Languages: The Case of BPEL4WS. *Conceptual Modeling - ER 2003*, *Lecture Notes in Computer Science*, vol. 2813. Springer Berlin / Heidelberg, 2003; 200–215, doi:http://dx.doi.org/10.1007/978-3-540-39648-2_18.

Prepared using speauth.cls
DOI: 10.1002/spe

18. Smith H. Business process management–the third wave: business process modelling language (bpml) and its pi-calculus foundations. *Information and Software Technology* 2003; **45**(15):1065 – 1069, doi:DOI:10.1016/S0950-5849(03)00135-6. Special Issue on Modelling Organisational Processes.
19. Arkin A, Askary S, Fordin S, Jekeli W, Kawaguchi K, Orchard D, Pogliani S, Riemer K, Struble S, Nagy PT, *et al.*. Web Service Choreography Interface (WSCI) 1.0. *Technical Report* 2002.
20. OASIS. Web Services Business Process Execution Language (WS-BPEL) version 2.0,. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html 2007.
21. KObjectsorg. Ksoap 2. http://ksoap2.sourceforge.net/.
22. KObjectsorg. kxml. http://kxml.sourceforge.net/.
23. Slominski A. XML Pull Parser (XPP). http://www.extreme.indiana.edu/xgws/xsoap/xpp/.
24. Pashtan A. *Mobile Web Services*. Cambridge University Press, 2005.
25. Farley P, Capp M. Mobile Web Services. *BT Technology Journal* 2005; **23**:202–213, doi:http://dx.doi.org/10.1007/s10550-005-0042-1. 10.1007/s10550-005-0042-1.
26. Tosic V, Lutfiyya H, Tang TY, Sherdil K, Dimitrijevic A. On requirements for management of mobile XML Web services and a corresponding management system. *Telecommunications in Modern Satellite, Cable and Broadcasting Services, 2005. 7th International Conference on* 28-30 Sept 2005; :57–60doi:10.1109/TELSKS.2005.1572063.
27. Gehlen G, Pham L. Mobile Web services for peer-to-peer applications. *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE* 3-6 Jan 2005; :427–433doi:10.1109/CCNC.2005.1405210.
28. Open Mobile Alliance Web site. http://ksoap2.sourceforge.net/.
29. Srirama SN, Jarke M, Prinz W. Mobile Web Service Provisioning. *Advanced International Conference on Telecommunications / Internet and Web Applications and Services, International Conference on* 2006; **0**:120, doi:http://doi.ieeecomputersociety.org/10.1109/AICT-ICIW.2006.215.
30. Zaplata S, Dreiling V, Lamersdorf W. Realizing Mobile Web Services for Dynamic Applications. *Software Services for e-Business and e-Society*, *IFIP Advances in Information and Communication Technology*, vol. 305, Godart C, Gronau N, Sharma S, Canals G (eds.). Springer Boston, 2009; 240–254, doi:http://dx.doi.org/10.1007/978-3-642-04280-5_20. 10.1007/978-3-642-04280-5_20.
31. Wolff A, Michaelis S, Schmutzler J, Wietfeld C. Network-centric Middleware for Service Oriented Architectures across Heterogeneous Embedded Systems. *Enterprise Distributed Object Computing Conference Workshops, IEEE International* 2007; **0**:105–108, doi:http://doi.ieeecomputersociety.org/10.1109/EDOCW.2007.20.
32. Kim YS, Lee KH. A lightweight framework for mobile web services. *Computer Science - Research and Development* 2009; **24**:199–209, doi:http://dx.doi.org/10.1007/s00450-009-0091-7. 10.1007/s00450-009-0091-7.
33. Juszczyk L, Dustdar S. A middleware for service-oriented communication in mobile disaster response environments. *Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing*, MPAC '08, ACM: New York, NY, USA, 2008; 37–42, doi:http://doi.acm.org/10.1145/1462789.1462796. URL http://doi.acm.org/10.1145/1462789.1462796.

Prepared using speauth.cls
DOI: 10.1002/spe

34. Schmidt H, Köhrer A, Hauck FJ. SoapME: a lightweight Java ME web service container. *Proceedings of the 3rd workshop on Middleware for service oriented computing*, MW4SOC '08, ACM: New York, NY, USA, 2008; 13–18, doi:http://doi.acm.org/10.1145/1462802.1462805.

35. Park G, Kim S, Bae G, Kim Y, Kang B. An Automated WSDL Generation and Enhanced SOAP Message Processing System for Mobile Web Services. *ITNG '06: Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06)*, IEEE Computer Society: Washington, DC, USA, 2006; 388–387, doi: http://dx.doi.org/10.1109/ITNG.2006.27.

36. Lai K, Phan TA, Tari Z. Efficient SOAP Binding for Mobile Web Services. *LCN '05: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, IEEE Computer Society: Washington, DC, USA, 2005; 218–225, doi:http://dx.doi.org/10.1109/LCN.2005.62.

37. Schall D, Aiello M, Dustdar S. Web services on embedded devices. 2006; 45–50, doi:http://dx.doi.org/10.1108/17440080680000100.

38. Kikkert S. Performance of Web Services on Mobile Phones. *Technical Report*, University of Groningen 2010.

39. AlShahwan F, Moessner K. Providing SOAP Web Services and RESTful Web Services from Mobile Hosts. *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, 2010; 174 –179, doi:10.1109/ICIW.2010.33.

40. Hassan M, Zhao W, Yang J. Provisioning Web Services from Resource Constrained Mobile Devices. *Cloud Computing, IEEE International Conference on* 2010; **0**:490–497, doi:http://doi.ieeecomputersociety.org/10.1109/CLOUD.2010.30.

41. Glombitza N, Pfisterer D, Fischer S. LTP: An Efficient Web Service Transport Protocol for Resource Constrained Devices. *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*, 2010; 1 –9, doi:10.1109/SECON.2010.5508255.

42. WS4D. What is DPWS and uDPWS all about? http://code.google.com/p/udpws/wiki/IntroductionGeneral#The_Internet October 2010.

43. Truong HL, Juszczyk L, Bashir S, Manzoor A, Dustdar S. Vimoware - A Toolkit for Mobile Web Services and Collaborative Computing. *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*, 2008; 366 –373.

44. Hackmann G, Gill C, Roman GC. Extending BPEL for Interoperable Pervasive Computing. *Pervasive Services, IEEE International Conference on*, 2007; 204–213, doi:10.1109/PERSER.2007.4283918.

45. Battista D, de Leoni M, De Gaetanis A, Mecella M, Pezzullo A, Russo A, Saponaro C. ROME4EU: A Web Service-Based Process-Aware System for Smart Devices. *Service-Oriented Computing - ICSOC 2008*, *Lecture Notes in Computer Science*, vol. 5364, Bouguettaya A, Krueger I, Margaria T (eds.). Springer Berlin / Heidelberg, 2008; 726–727, doi:http://dx.doi.org/10.1007/978-3-540-89652-4_65.

46. van Engelen RA, Gallivan K. The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. *2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002)*, 2002; 128–135.

Prepared using speauth.cls
DOI: 10.1002/spe

47. uDPWS. The Device Profile for Web Services (DPWS) for highly resource-constrained devices. http://code.google.com/p/udpws/.

DOI: 10.1002/spe