Enabling Resource-oriented Mobile Web Server for Short-Lived Services

Fahad Aijaz, Syed Zahid Ali, Muzzamil Aziz Chaudhary, Bernhard Walke
Department of Communication Networks
Faculty 6, RWTH Aachen University
Kopernikusstr. 5, 52074 Aachen, Germany
Email: fah@comnets.rwth-aachen.de

***Abstract*—Mobile Web Services (MobWS) are a promising technology for the integration of Information Technology (IT) and Telecommunication (TelCo) domains. The advancements in mobile communication and device technology, in terms of software and hardware, have revolutionized the focus of research and industry to investigate mobile service provisioning platforms and architectures.**

In this paper, we present a concept and architecture of a REST-Interfaced Mobile Web Server as a service delivery platform. Here, we discuss about the interaction strategy and URL-based access techniques for short-lived synchronousMobWS. The paper focuses on in-depth explanation of software architecture and its components of server and compares the payload requirements with the state-or-the-art SOAP based MobWS. The REST architecture has significantly reduced payload demands that shows promising signs of optimized processing performance of the mobile server when compared to SOAP.

I. Introduction

In recent years, WSs specified by the World Wide Web Consortium (W3C) have evolved as a highly-regarded implementation of SOA for system integration. The Internet and mobile networks have started to merge based on WSs spotlighting high-valued consumer and business services, and use cases. The combination of mobile communication systems and WSs foresee high economical potential in terms of WS provisioning from a mobile node, called MobWS. From research and technological perspective, the maturity process is ongoing; however several challenges must be addressed to enable integrated P2P MobWS systems of IT and TelCo domains.

The global acceptance of MobWSs as a mature platform is still in its infancy. Enabling the true potential of MobWSs in the current highly competitive global market faces several research challenges and requires recommendations for efficient solutions at various technical levels. [1] presents a detailed study, based on the experience of several research projects, about challenges for developing middleware on smart phones. The importance of resource management, lightweight communication protocols and asynchronous programming are also highlighted. The work in [2], on the other hand, develops a Mobile Host platform for provisioning MobWSs. In this work, in order to reduce the processing latencies within the Mobile Host, BinXML compression technique is adapted. The research in [3] presents the concept of first ever Mobile Web Server and comprehensively analyzes the traffic performance characteristics of MobWS. It further classifies MobWS into three distinct classes, MobWS Access, MobWS Provisioning and P2P MobWS, and presents multiple transport protocol bindings for SOAP. Within the scope of research in [3], the mobility issues in MobWS domain are also addressed. Recently a technical report from Nokia Research Center in Helsinki presented the concept of providing HTTP access to Web Servers running on mobile devices [4]. It is further described by the same group in [5], the technical approach of porting the Apache httpd to the Symbian/S60 mobile platform in order to enable Website hosting and access on mobile phones.

This work has been supported by the UMIC Research Center, RWTH Aachen University.

In this paper we intend to study the architectural for the P2P MobWS enfolding short-lived synchronous communication. In the first phase, synchronous MobWS interaction strategy is discussed based on the operations specified in Web Service Description Language (WSDL) standard [6]. Based on these grounds, the synchronous server-side architectural components are discussed in detail, taking into account the Representational State Transfer (REST) and Simple Object Access Protocol (SOAP) messaging frameworks. The later phase evaluates and compares the influence on the architectural performance of the server caused by each access technique. Some preliminary performance evaluation is conducted based on payload optimization and real-time measurements depicting optimized processing.

II. Architecture of a REST-Interfaced Mobile Web Server

The MSWs, due to their instantaneous nature, simplify the design process of their underlying architecture. The architecture strongly relies upon the Synchronous Interaction (SI) discussed earlier. Earlier research in [3] presents a comprehensive architecture and research findings in the area of Service Oriented Architecture (SOA) based MSWs. There, the SOAP standard is used to define the Messaging Framework (MF) and presents several Mobile Synchronous Web Services (MSW) access mechanisms over multiple transport protocols. Detailed performance evaluation and theoretical analysis in terms of measurements and analytical models are also presented. Within the scope of this research, we extend the work in [3] to support Resource Oriented Architecture (ROA)-based MSWs, called 'RESTful MSWs (R-MSW)', using REST design principles. Here we do not discuss the MSW architecture based on SOA reference model; however we do present its comparison with REST in terms of message payloads.

A. *Synchronous Interaction*

The SI is suitable for systems performing short-lived tasks. A P2P MobWS provisioning system directly inherits the properties of standard WWW and the Internet. Thus, the MobWSs that are designed to perform short-lived P2P operations strongly integrate SI as their underlying communication pattern. These MobWSs are termed as MSW as depicted in Figure 1. The figure illustrates a simple scenario when one peer initiates the request-response process in order to consume the service provisioned by the other peer.

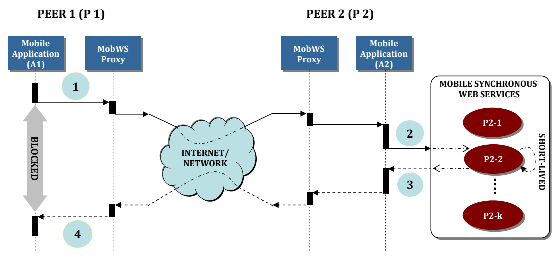


Fig. 1. P2P Synchronous Interaction

MSWs are suitable for use cases demanding immediate response to a request. This is possible by ensuring the completion of service operation within a small time frame. Generally, MSW are widely realized in an RPC-oriented manner; however other techniques are also possible.

B. *Uniform Resource Locator (URL) Defined Synchronous Access*

Migrating a system from SOA to REST architecture is not straight forward and raises several design requirements. The Extensible Markup Language (XML) based MobWS implementation of SOA is highly flexible and extensible toward heterogeneous networks and changes due to the transport neutral behavior of SOAP and variety of existing Web Services (WS) standard specifications. This however leads to coarse-grained and thick SOAP messaging structures, especially in case of mobile terminals. Realizing a system based on the REST design principles imposes a dependency on Hypertext Transfer Protocol (HTTP) and URL standards, while significantly reducing the message payloads. The existing WS standard specifications can be optimized for REST architecture in order to handle requirement changes.

Exposing MobWS resources, defined by service methods, with REST demands a clear definition of URL structure that must be used to access R-MSW resources. In order to meet this requirement, we develop generic URL structures for synchronous interaction as shown in Figure 2.

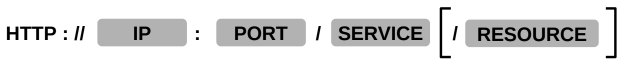


Fig. 2. URL Structure Enabling Synchronous Access

Due to the strict dependency of REST on HTTP, binding to other transport protocols is not recommended since in such case, several design criterion have to be compromised. The R-MSWs are instantaneous services that rely solely on the behavior of SI. Thus, in order to consume such services, the requester must formulate a URL, published by the service provisioning node, which conforms to the structure shown in the figure. The URL of the serving node consist of an Internet Protocol (IP) address and the listening PORT. Since the resources of R-MSWs are directly accessed, therefore the SERIVCE parameter represents the name of the service that the request intends to consume. Unlike SOAP request, the service name is not specified in the HTTP payload. The specification of service name in the URL helps in avoiding the overheads of SOAP parsing and to identify the invoked service. Thus, the R-MSWs resource is directly exposed on the Internet/network. For instance, a simple R-MSWs 'PublicationService', which maintains the publications records on the peer node, can be directly accessed using the URL http://rest.comnets.de:9090/PublicationService. Even a single R-MSW may perform variety of operations, such as get, update, insert or delete. In such cases, the URL alone does not specify the actions that a requester intends to take.

C. *Mapping HTTP Methods to URL*

Due to the dependency on HTTP, the REST architectural principles are strictly coupled with the HTTP methods to convey the intended actions of the requester. The HTTP has a set of methods with predefined goals [7], however, the most commonly used among them are GET, POST, PUT and DELETE. Mapping the HTTP URL to the HTTP methods, facilitates indicating the purpose of clients' request. In the Publication-Service example, mapping of the service URL to the HTTP methods entirely changes the context of request. For example, if the URL http://rest.comnets.de:9090/PublicationService is mapped to the HTTP GET method, the service only provides the list of publications. Mapping to HTTP POST represents an update action and the service modifies the record of an existing publication. The PUT method focuses on creating new information and can be used to perform an insert operation, like adding new publication to the records. Similarly, the URL mapping to DELETE method signifies the removal of a publication from the list that the PublicationService has maintained. In situations where a service offer multiple resources of one kind, such as various getX() methods like getList, getAuthors, getConferences etc., a simple mapping of URL http://rest.comnets.de:9090/PublicationService to HTTP GET does not convey the targeted action of the request. Thus, the optional parameter for specifying the target RESOURCE must be used. With this extension, the requester can directly specify the target resource in the URL and map to the corresponding HTTP method. For example, in order to fetch the list of publication, the requester formulates the URL http://rest.comnets.de:9090/PublicationService/List and maps it to the HTTP GET method, which clearly indicates the targeted resource and action; getList operation. In order to update, insert or delete the List, the same URL is used with its respective mappings to POST, PUT and DELETE methods.

D. Server-side Architectural Components

The system proposed in [3] uses SOAP Remote Procedure Call (RPC) mechanism to consume SOAP interfaced MobWS. Thus, any SOAP request uses *almost* the same URL structure as shown in Figure 2. The difference is created by replacing the SERVICE parameter in the URL structure with *soaprpc* and the RESOURCE parameter is completely removed. The *soaprpc* parameter is specific to SOAP architecture and is required to identify a SOAP RPC call. Since every SOAP call must carry a SOAP envelope in HTTP payload, therefore POST HTTP method is utilized.

On the other hand, any request for consuming MobWS over the REST interface conforms to the URL structures presented in Section II-B. Therefore, for any REST request, the parameter *soaprpc* is not required which eases the request type identification process by relying upon the structure of received URL. Hence, the Listener component after receiving the request, identifies the target architecture by checking for the *soaprpc* parameter in the HTTP URL.

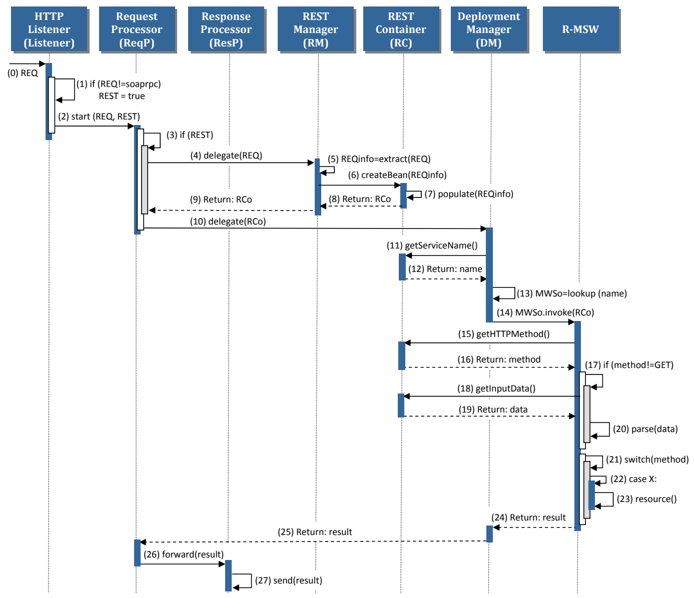


Fig. 3. Sequence Diagram of the Server-side Architectural Components

Figure 3 illustrates the invocation process of R-MSW in server architecture in a mobile terminal. The flow of information and control transitions between different server components of REST architecture is shown. Upon arrival of client's request REQ, the HTTP Listener component (Listener) identifies the request type based on the soaprpc parameter, as explained previously. Assuming the request for R-MSW, the Listener initiates a thread called Request Processor (ReqP) by providing REQ and request type identification flag, in this case, REST. The ReqP is a parent server component responsible for managing incoming requests in accordance to their requirements by delegating and initializing other server components on demand. Since the REQ targets the R-MSW, the ReqP simply delegates the control flow to REST Manager (RM) after checking the REST flag. The RM is a generic and major server component specially designed, as an extension to original architecture, for MobWSs based on REST design principles. The responsibility of RM is to extract all the information embedded within the REST REQ and create a read-only container namely, REST Container (RC). The RC is a de-serialized version of REQ that is understandable by all the server components involved in the invocation process of R-MSW. Before the RC is created, the RM has to perform variety of operations such as, parsing the URL and understanding its structure, obtaining the HTTP Method, extracting the input data from HTTP Payload and check for URL faults. Assuming a no-fault scenario, the RM creates a coarse-grained RC that populates all the extracted information in its properties. The read-only nature of RC facilitates in strictly conforming to the REQ, by preventing any server component or service to accidentally modify the contained information. The REST Container object (RCo) of RC is returned to the ReqP by RM once it is created. Since the information delivered by the client in REQ is now encapsulated within the RCo, therefore its reference can be passed around as a single object across other server components instead of multiple individual objects. Upon receiving the RCo from RM, the ReqP delegates the control flow, along with the RCo, to Deployment Manager (DM) component which is responsible for lookup and invocation of the requested MobWS. The DM uses the RCo to obtain the requested service name from RC. Since in DM, a list of available services along with their corresponding instances is maintained as a key-value mapping, therefore DM uses the R-MSW name, obtained from RC, in order to lookup the related MobWS object (MWSo). The MWSo is then used to invoke the R-MSW and RCo is provided as an argument.

At this phase, the invocation of R-MSW does not imply invocation of the requested service method in REQ. The R-MSW is responsible for evaluating the mapping between the HTTP method and the URL in order to identify the targeted service method from the client. For this reason, R-MSW obtains the HTTP method from RC through the received RCo. Upon receiving the method, the service may obtain the target resource using RCo, if desired. Obtaining the target resource is only necessary if the R-MSW offers multiple methods of same kind, for e.g. many getX() methods. For services offering only one method of each kind, the mapped HTTP method can be used directly for target invocation. Since providing multiple or single kinds of methods is specific to service requirements, therefore for keeping the discussion generic, call to obtain target resource from RC is not shown in the figure. In the next step, the R-MSW must check if the client has provided some input data that should be used by the target resource.

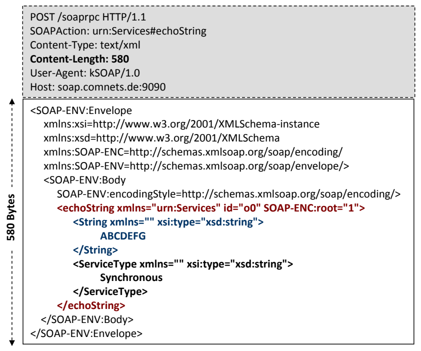
Since the URL mapping to HTTP GET method implies no payload in REQ, therefore the service only requests the input data from RC (using RCo) if and only if the HTTP method other than GET is mapped. In case the payload exists, the R-MSW parses the input data and subsequently the target service method is invoked which is directly identified by the HTTP method mapping (illustrated as case X). Consequently, the target R-MSW resource, upon completion, dispatches the result which is received by the parent server component ReqP through the DM. The ReqP forwards the result to Response Processor (ResP) component which sends the result to the client using the same connection.

III. Payload Optimization

Independent of the underlying architecture, synchronous MobWS are developed to be a short-lived processes. Therefore, a synchronous requests carries the input data for the invoked service as an HTTP payload. In case of SOA, this payload is embedded within the SOAP envelope, whereas with REST architecture, it is embedded in a format that conforms to the application requirements. Since in REST architecture, a resource can be directly identified by the URL, therefore extensive SOAP parsing can be avoided.

In order to evaluate the payload optimization with REST, we developed a simplest possible synchronous MobWS called '`echoString`' and hosted it on the MobWS framework. The sole function of the `echoString` service was to receive a string input parameter from the client and echo it back as a response. The service was exposed for both REST and SOAP MobWS architectures in order to study the message constructs and payloads for each type of invocation. The SOAP request was constructed with a well-known third party library for Java Micro Edition (Java ME) called kSOAP, whereas for REST request standard Java ME API was utilized. In Figure 4(a) and 4(b) the corresponding SOAP and REST MobWS requests are depicted.

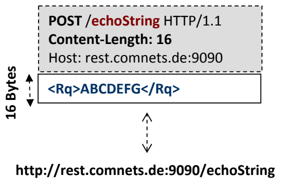
In comparison to SOAP, significant reduction of $\approx 98\%$ in MobWS request payload be easily observed in case of REST. By using the URL structure for R-MSW acess, as illustrated in Figure 2, the contents of SOAP request can directly be mapped to the REST URL in order to avoid XML parsing latencies. For instance, the `<echoString ...>` element in the SOAP message is mapped to URL parameter echoString which is present in the HTTP headers of REST request. Similarly, the `<String ...>` element that carries the input data to service is completely replaced by the HTTP payload of REST request. With the predefined synchronous URL structure, we were able to avoid the use of `<ServiceType ...>` element (refer to [8]). Due to the existance of payload in REST request, the URL was mapped to the POST method of HTTP. Since the synchronous MobWS are based on SI, therefore the response from the `echoString` was also transported over the same HTTP connection. The responses received from the invoked MobWS in case of SOAP and REST are depicted in Figure 5(a) and 5(b) respectively.



```
POST /soaprpc HTTP/1.1
SOAPAction: urn:Services#echoString
Content-Type: text/xml
Content-Length: 580
User-Agent: kSOAP/1.0
Host: soap.comnets.de:9090
```

```
<SOAP-ENV:Envelope
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xmlns:xsd=http://www.w3.org/2001/XMLSchema
   xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
   xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/>
   <SOAP-ENV:Body
      SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/>
      <echoString xmlns="urn:Services" id="o0" SOAP-ENC:root="1">
         <String xmlns="" xsi:type="xsd:string">
               ABCDEFG
         </String>
         <ServiceType xmlns="" xsi:type="xsd:string">
               Synchronous
         </ServiceType>
      </echoString>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(a) Synchronous SOAP Request for echoString MobWS



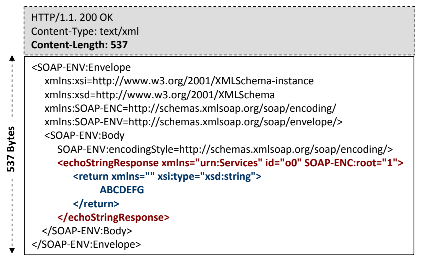
```
POST /echoString HTTP/1.1
Content-Length: 16
Host: rest.comnets.de:9090
```

```
<Rq>ABCDEFG</Rq>
```

16 Bytes
http://rest.comnets.de:9090/echoString

(b) Synchronous REST Request for echoString MobWS

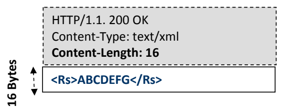
Fig. 4. HTTP Payloads for Synchronous MobWS Request



```
HTTP/1.1. 200 OK
Content-Type: text/xml
Content-Length: 537
```

```
<SOAP-ENV:Envelope
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xmlns:xsd=http://www.w3.org/2001/XMLSchema
   xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
   xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/>
   <SOAP-ENV:Body
      SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/>
      <echoStringResponse xmlns="urn:Services" id="o0" SOAP-ENC:root="1">
         <return xmlns="" xsi:type="xsd:string">
               ABCDEFG
         </return>
      </echoStringResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(a) Synchronous SOAP Response from echoString MobWS



```
HTTP/1.1. 200 OK
Content-Type: text/xml
Content-Length: 16
```

```
<Rs>ABCDEFG</Rs>
```

(b) Synchronous REST Response from echoString MobWS

Fig. 5. HTTP Payloads for Synchronous MobWS Response

Since the same HTTP connection is used for the response messages, therefore the requirement of explicitly transmitting the `<ServiceType...>` element in case of SOAP is no longer relevant, whereas no URL is generated for the REST response. Compared to SOAP, significant HTTP payload reduction of $\approx 97\%$ is achieved in synchronous response with the REST MobWS provisioning architecture. With further architecture evaluations, a promising performance improvement in terms of processing latencies is expected due to the exponential reduction in payload. Some priliminiary performance results in terms of processing latencies in SOAP and REST Mobile Web Server architectures are compared in Figure 6.

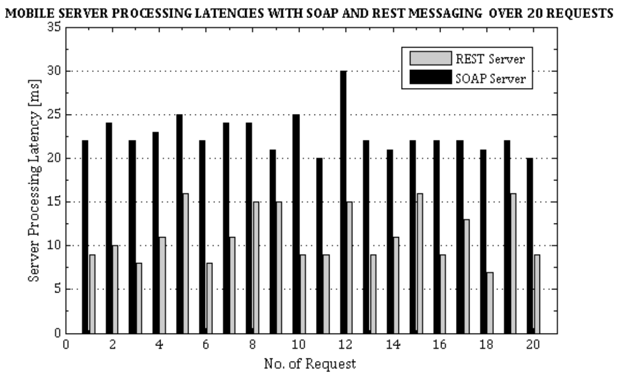


Fig. 6. Preliminary Performance Comparison of SOAP and REST Mobile Web Servers

IV. Conclusion

In this paper, a concept and architecture of a REST-Interfaced Mobile Web Server is presented with a strong focus towards MobWS provisioning. In the first phase, synchronous interaction strategy for short-lived MobWS is elaborated. Based on these grounds, the architecture, based on REST design principles, is discussed in detail considering the server-side architectural components. It has been seen that the payload demands are significantly reduced with REST architecture for Mobile Web Server when compared to SOAP which prominently effects the processing latencies and shows optimized results.

References

[1] O. Riva and J. Kangasharju, "Challenges and lessons in developing middleware on smart phones," *IEEE Computer Magazine*, October 2008.

[2] S. N. Srirama, "Mobile hosts in enterprise service integration," Ph.D. dissertation, RWTH Aachen University, 2008. [Online]. Available: http://darwin.bth.rwth-aachen.de/opus/volltexte/2008/2567/

[3] G. Gehlen, "Mobile web services - concepts, prototype, and traffic performance analysis," Ph.D. dissertation, RWTH Aachen University, Lehrstuhl fr Kommunikationsnetze, Aachen, Germany, Oct 2007. [Online]. Available: http://www.comnets.rwth-aachen.de

[4] F. D. Johan Wikman, "Providing http access to web servers running on mobile phones," Nokia Research Center Helsinki, Tech. Rep. NRC-TR-2006-005, May 2006. [Online]. Available: http://research.nokia.com/tr/NRC-TR-2006-005.pdf

[5] M. T. Johan Wikman, Ferenc Dosa, "Personal website on a mobile phone," Nokia Research Center Helsinki, Tech. Rep. NRC-TR-2006-004, May 2006. [Online]. Available: http://research.nokia.com/tr/NRC-TR-2006-004.pdf

[6] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," Published on the internet, May 2005, w3C Recommendation. [Online]. Available: http://www.w3.org/TR/2005/WD-wsdl20-20050510

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," United States, 1999.

[8] F. Aijaz, H. Bilal, and W. Bernhard, "Asynchronous mobile web services: Concept and architecture," in *Proceedings of the IEEE 8th International Conference on Computer andInformation Technology*. Sydney, Australia: IEEE, July 2008, p. 6. [Online]. Available: http://www.comnets.rwth-aachen.de