# Enabling High Performance Mobile Web Services Provisioning

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, zhd@comnets.rwth-aachen.de

***Abstract*—The Telecommunication (TelCo) and Information Technology (IT) domains are converging at their intersection to provide vast variety of efficiently accessible services to the consumers. Specially in mobile communication domain, a vital factor that can affect the performance of any mobile application is the resource constraint nature of mobile devices with limited processing and storage capacity. On the other hand integration of heterogeneous networks and systems bring influential research challenges.**

**In this paper, we present a concept of REST based Mobile Web Services (MobWS) provisioning and its comparison with a similar SOAP architecture in terms of HTTP payload. The requirements of REST architecture are discussed and URL structures are defined for the interaction with synchronous and asynchronous RESTful MobWS. A resource oriented approach for optimizing HTTP payload is presented and related message structures are discussed. It is shown that the REST based MobWS provisioning when compared to SOAP, offers promising results by significantly reducing the payload.**

## I. Introduction

Service Oriented Architecture (SOA) [13] is a widely applied standard reference model for service oriented computing, with Web Services (WS) being its most common realization. Today, WS are not only limited to fixed servers but can be provisioned from a mobile device. These WS are termed as MobWS. A MobWS system realization based on SOA encounter high payloads due to extensive parsing requirements of Simple Object Access Protocol (SOAP) and its extensions. The everyday increasing WS specification standards bring new performance challenges for MobWS provisioning node due to increasingly thick message payloads.

On the other hand, Representational State Transfer (REST) also termed as Resource Oriented Architecture (ROA), is a software architecture style for distributed hypermedia systems such as the World Wide Web (WWW) [4]. A REST system supports direct interaction with WS based on the standard Uniform Resource Locator (URL) [2] that facilitates avoiding application performance degrading factors such as SOAP parsing in SOA. Systems based on REST are tightly coupled with Hypertext Transfer Protocol (HTTP) [3].

The research community has produced several results in order to facilitate decision process between SOA and REST architectures for enterprise and mobile computing [14], [15].

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

In this paper, a MobWS provisioning system based on the REST architectural principles is presented that avoids the overheads of SOA based systems in terms of HTTP payloads and interaction mechanisms.

## II. Mobile Web Service Provisioning

The concept of provisioning WS from a mobile node was first introduced in [10] by presenting a SOAP server as a MobWS provisioning middleware. Since then, comprehensive research activities have been conducted by the same group that identifies the classes, use cases and transport performance overheads of MobWS [6]–[9], [11]. The concrete and global acceptance of MobWS provisioning concept demands efficient solutions not only for high performance transport protocols and networks, but also for software architecture of the middleware that enables hosting WS on a mobile node.

In this Section we present a concept of a well-known REST style [4] applied to achieve an efficient MobWS provisioning architecture by avoiding the overheads of thick SOAP message payloads while using standard Internet technologies, such as HTTP and URL.

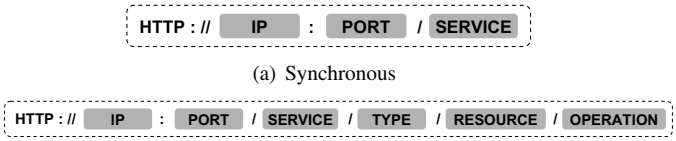
### A. RESTful Mobile Web Services

WS that conforms to the REST architectural principles are known as 'RESTful WS'. Such WS when provisioned from a mobile node are termed as 'RESTful MobWS'. Research in [1] has shown that the interaction with MobWS can be short- or long-lived, and are referred as synchronous or asynchronous respectively. Within the scope of this work, an integrated architecture for RESTful MobWS provisioning is presented for each type of interaction mechanism.

*1) Requirements:* 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 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 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.

*a) URL Structure:* Exposing MobWS as resources with REST demands a clear classification of distinct URL for each type of service interaction i.e. synchronous and asynchronous. In order to meet this requirement, we develop two distinct generic URL structures for each interaction type as shown in Figure 1(a) and 1(b). Each of the two URL structures are explain in the following description:




(a) Synchronous

(b) Asynchronous

Fig. 1. URL Structures for RESTful MobWS Interactions

1) SYNCHRONOUS STRUCTURE: Synchronous MobWS are simple request-response processes that keep a client in a blocked state until the response has arrived. In order to consume such services, the only requirement is to identify the service provisioning node in terms of its Internet Protocol (IP) (or domain name) and listening port. Figure 1(a) depicts this behavior of synchronous MobWS interaction. The HTTP is used as a standard protocol that conforms to the REST architecture style [4], whereas the IP and listening port of the service provider must be known by the client. The SERVICE in Figure 1(a) corresponds to the name of synchronous MobWS that the client intends to consume. The MobWS provisioning node must be capable of invoking the service instance based on the name provided by the client in the incoming synchronous URL. An example URL for synchronous RESTful MobWS is shown in Figure 2.

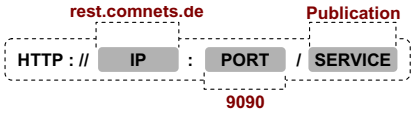



Fig. 2. REST URL for Synchronous MobWS

2) ASYNCHRONOUS STRUCTURE: Unlike synchronous MobWS, asynchronous services are complex processes enabling variety of communication strategies. Due to the long-lived behavior of asynchronous MobWS, their interaction could start with a simple request-response process and later demand complex callback or polling based service feedbacks. In either scenario, it is important to precisely identify the resource and its operation within the REST architecture of the provisioning node that a client intends to invoke. The service provider must be able to distinguish the synchronous request from asynchronous request and delegate the control flow to the corresponding component of the architecture. In Figure 1(b), it can be observed that the first three parameters of the asynchronous URL structure are identical to the synchronous structure (See Figure 1(a)), however, the new requirements arise after this point. The 'TYPE' parameter in the asynchronous structure corresponds to the type of MobWS request. For instance, for the asynchronous request, the value of TYPE would be 'Asynchronous'. For the sake of simplicity, we do not add TYPE parameter to the synchronous URL structure and the request is implicitly assumed to be of type synchronous if nothing succeeds the SERVICE parameter. The RESOURCE parameter is critical to the system and represents the architectural component the asynchronous request should be delegated to. For instance, in case of asynchronous MobWS invocation request, the control flow is delegated to the `Factory` component, while for control or monitoring request, the `Instance` component is responsible (see Figure 4). Each of these components provide a set of functions that are exposed as lowest-level of accessible resources in our system. In the URL structure of Figure 1(b), we represent these functions as OPERATION. Having defined the URL structure for asynchronous MobWS, an example URL for service creation is depicted in Figure 3.

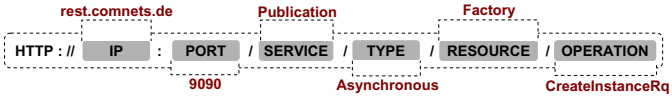



Fig. 3. REST URL for Asynchronous MobWS

In order to understand the underlying architectural components and their corresponding functions in the asynchronous MobWS provisioning system, a study of [1], [5] is highly recommended.

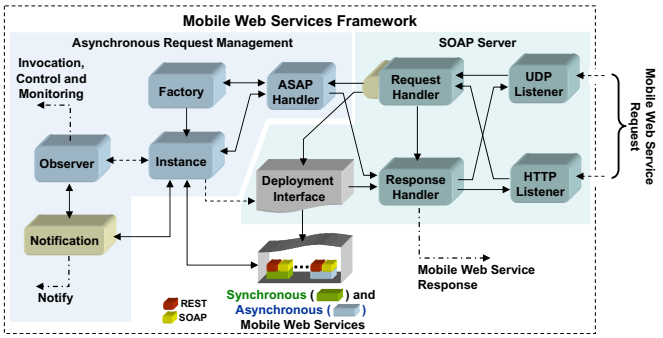



Fig. 4. Mobile Web Services Provisioning Architecture

*b) HTTP Methods:* Due to the dependency on HTTP, the REST architectural principles are strictly coupled with the HTTP methods. HTTP has a set of methods with predefined goals [3], 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. Within the scope of this work, we map the GET and POST methods of HTTP to the REST URL structures in order to directly depict the intended behavior of the request, roughly categorized as (a) READ, and (b) CREATE.

Any request that does not carry any HTTP payload and demands retrieval of information without changing the behavior of the system is seen as a READ request. In this case, every REST URL is mapped to the GET method of HTTP. For instance, a MobWS client may ask for current properties of the Publication service in order to analyze further request decisions. The combined affect created by the mapping of URL to HTTP method clearly identifies the purpose of request, since the accessed resource and its operation is represented by the URL and the HTTP method depicts the actions the resource or its operation shall perform. An example asynchronous REST URL mapped to HTTP GET method to obtain the service properties is shown in Figure 5.

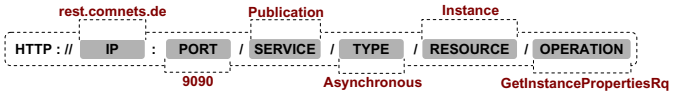



Fig. 5. REST URL for Asynchronous MobWS GET Request

The CREATE requests carry input data as an HTTP payload or changes the system behavior. Due to short-lived request-response nature, the synchronous MobWS does not support interactions during runtime and the client remains in the blocked state. Therefore only type of request for such MobWS is to directly invoke the service that causes a change in system behavior. The input data to these MobWS is supplied as an HTTP payload. Since these requests are categorized as CREATE request, therefore only HTTP POST method can be mapped to the synchronous REST URL. Thus, when the example URL shown in Figure 2 is mapped to POST HTTP method, a synchronous MobWS request is represented. The CREATE requests for asynchronous MobWS support variety of operations provided by the resources of the MobWS provisioning architecture (refer to [1], [5]). One example is presented in Figure 3, since the service creation process requires the input data as HTTP payload that changes the system behavior by initializing corresponding new MobWS objects and threads in the architecture. Here, we do not discuss all possible CREATE requests for asynchronous MobWS, however for basic understanding, Table I lists some resource operations and their mapping to HTTP methods. The complete list of operations can be obtained from [5].

### B. MobWS Messaging with SOAP and REST

The design of an integrated MobWS provisioning architecture supporting short- and long-lived MobWS with variable interactions is a key factor influencing its performance. Not only the architectural components, but also the messaging and flow of control must be carefully analyzed to reduce processing latencies. In this section, we discuss the REST architecture message constructs for MobWS and compare them with SOAP based MobWS system. Considering the requirements of REST

| Resource Operations | HTTP Methods |
|---|---|
| GetFactoryPropertiesRq | GET |
| CreateInstanceRq | POST |
| ChangeState | POST |
| Completed | POST |
| Synchronous MobWS | POST |

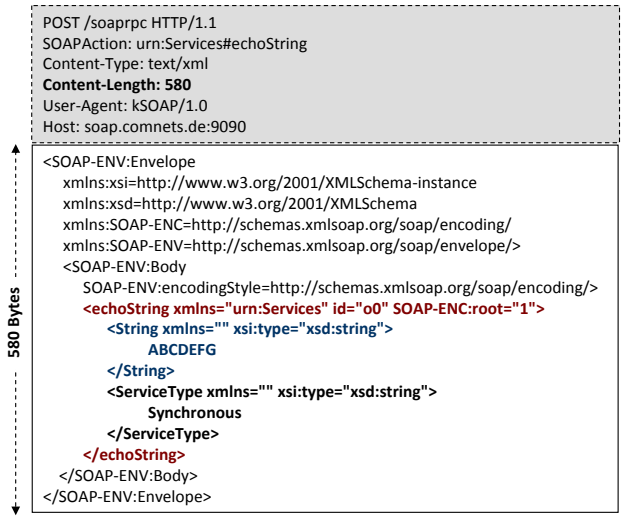
TABLE I
MAPPING OF OPERATIONS TO HTTP METHODS

architecture presented in Section II-A, we try to identify how the RESTful MobWS provisioning design reduces the HTTP payload when compared to SOAP based MobWS invocation for similar requests. Here, we focus on both, synchronous and asynchronous interaction of MobWS.

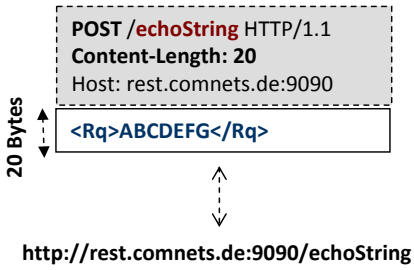
*1) Synchronous MobWS:* Independent of the underlying architecture, synchronous MobWS are developed to be a short-lived processes. Therefore, a synchronous requests carries the input data for the invoked service as an HTTP payload. In case of SOA, this payload is embedded within the SOAP envelope, whereas 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 that is required for invoking a service.

In order to prove our claims, we developed a simplest possible synchronous MobWS called '`echoString`' and hosted it on our middleware platform [1]. 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 (see Figure 4). The SOAP request was constructed with a well-known third party library for Java Micro Edition (J2ME) called kSOAP, whereas for REST request standard J2ME API was utilized. In Figure 6(a) and 6(b) the corresponding SOAP and REST MobWS requests are depicted.

The significant reduction in MobWS request payload over HTTP can be easily observed. By using the REST URL structure for synchronous MobWS illustrated in Figure 1(a), 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. Due to the predefined synchronous URL structure, we were able to avoid the use of `<ServiceType ...>` element since the interaction type was directly identified by the URL. Due to the existence of payload in REST request (CREATE request, see Section II-A1), the URL was mapped to the POST

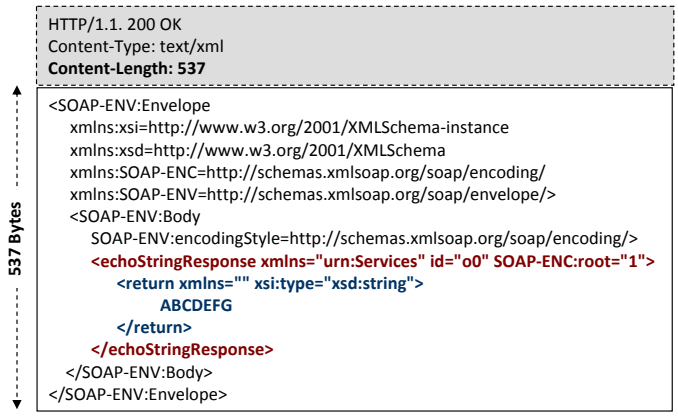



(a) Synchronous SOAP Request for echoString MobWS

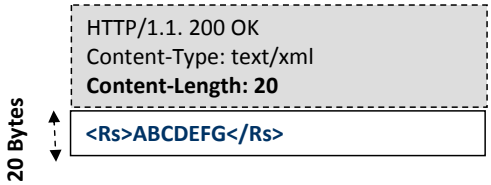



(b) Synchronous REST Request for echoString MobWS

Fig. 6. HTTP Payloads for Synchronous MobWS Request




(a) Synchronous SOAP Response from echoString MobWS




(b) Synchronous REST Response from echoString MobWS

Fig. 7. HTTP Payloads for Synchronous MobWS Response

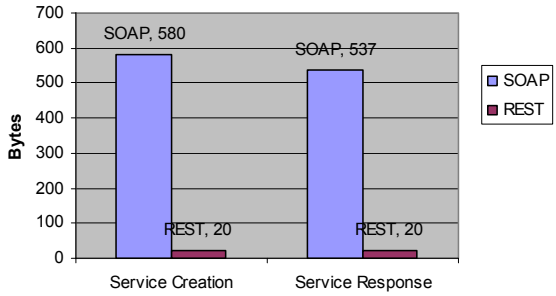



Fig. 8. Comparison of HTTP Payloads for Synchronous MobWS

method of HTTP. Since the synchronous MobWS request is a blocking call that keeps the client in a listening state, 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 7(a) and 7(b) respectively.

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 is achieved in synchronous request-response process 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. A comparison of the HTTP message payloads for SOAP and REST based synchronous MobWS request-response process is depicted in Figure 8.

*2) Asynchronous MobWS:* Unlike synchronous request-response process, the message constructs in asynchronous MobWS becomes complex due to incrementing requirements. This is because a single asynchronous MobWS does not only represents a request-response process, but also facilitates client interaction with the MobWS at later times in terms or control and monitoring. Thus, a demand arises for defining new message structures which are coupled with callback and polling interaction mechanisms.

Asynchronous MobWS supports variety of processes, such as service creation, invocation, control and monitoring. In synchronous system, MobWS creations and invocation process are not distinct actions, as there is no concept of interaction with service at later time. However, these processes when opted for asynchronous systems, must be clearly defined. Such architectures classify service creation as a separate process that does not necessarily involve a service invocation. Although, the definition of a service creation process may vary depending upon the context and architecture it is designed for, we specify service creation as a process of instantiating a service object such that it moves to a READY state as shown in Figure 9. A service can only be in a ready state once all the required information is provided that is enough for a service to perform its functions immediately or at some later time. The invocation process of an asynchronous service is time dependent which

usually starts after a certain predefined timer expires while the service is in the ready state. Within the scope of this work, we only consider immediate service invocation process, that is, the MobWS immediately transitions from READY to RUNNING state once the service creation process is finished.

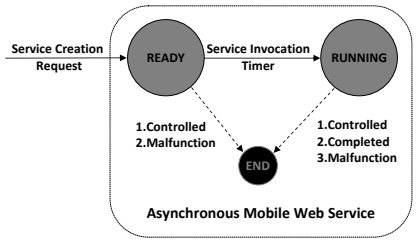



Fig. 9. States of Asynchronous MobWS

As in Figure 9, an asynchronous MobWS can reach the END state, either from READY or RUNNING states. From the READY state this is possible by sending a control message that terminates the service [1], [5] and also due to some internal malfunctioning of the system, such as memory conflict, thread deadlock etc. In addition to the control and malfunctioning scenarios, an asynchronous service in a RUNNING state can also transition to the END state once it has completed performing its tasks. This is not possible if only service creation process takes place (in READY state) since the service cannot complete its tasks unless it is started (in RUNNIGN state).

In order to meet these requirements, [5] defines the message constructs based on SOAP. However, for a mobile terminal, SOAP message constructs tends to be the performance degrading factor due to their thick message sizes. On the other hand, the REST architecture facilitates achieving the same goals with significantly reduced message payloads.

Following the similar approach as discussed in Section II-B1, we developed a simple asynchronous MobWS called `aEchoString` in order to study the relation between the SOAP and REST message payloads when transported over HTTP. The `aEchoString` MobWS served the same purpose as `echoString` service, but by following the asynchronous MobWS invocation process [1]. This entirely changes the message structure since it conforms to the specification in [5]. Firstly, we invoked the `aEchoString` MobWS with the SOAP interface by sending a service creation request and the immediate service invocation flag set to true. Figure 10 depicts the SOAP message for this request that was transmitted as an HTTP payload.

The invoked MobWS is identified by the `<aEchoString ...>` element and the information contained within the `<createInstanceRq ...>` element is the actual information required by the asynchronous MobWS to initiate the process [5]. The element also identifies the target operation `createInstanceRq` of the accessed resource 'Factory' represented by the `MethodType` element. The interaction type as 'Asynchronous' is identified by parsing the element `<ServiceType ...>`. As specified by [5], the SOAP

```
POST /soaprpc HTTP/1.1
SOAPAction: urn:Services#aEchoString
Content-Type: text/xml
Content-Length: 1559
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/
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
   <SOAP-ENV:Header>
      <wsa:To>http://soap.comnets.de:9090/soaprpc</wsa:To>
      <wsa:Action> urn:Services#aEchoString</wsa:Action>
      <wsa:MessageId>uuid:ffb471ec-194-1cb37664-125c53c2</wsa:MessageId>
      <wsa:ReplyTo>
         <wsa:Address>http://soap.comnets.de:40985/soaprpc</wsa:Address>
      </wsa:ReplyTo>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body  SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/>
      <aEchoString xmlns="urn:Services" id="o0" SOAP-ENC:root="1">
         <ServiceType xmlns="" xsi:type="xsd:string">Asynchronous</ServiceType>
         <MethodType xmlns="" xsi:type="xsd:string">Factory</MethodType>
         <createInstanceRq xmlns="" xsi:type=":createInstanceRq">
               <StartImmediatelyxsi:type="xsd:boolean">true</StartImmediately>
               <ObserverEPR xsi:type="xsd:string">
                        http://soap.comnets.de:40985/soaprpc
               </ObserverEPR>
               <Name xsi:type="xsd:string">aEchoString</Name>
               <Subject xsi:type="xsd:string">Echo String</Subject>
               <Description xsi:type="xsd:string">aEchoString Msg</Description>
               <ContextData xsi:type=":ContextData"><!-- 84 bytes --></ContextData>
         </createInstanceRq>
      </aEchoString>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1559 Bytes

Fig. 10. Asynchronous SOAP Service Creation Request for aEchoString MobWS

headers make use of WS Addressing [12] standard in order to exchange Endpoint Reference (EPR) between the client and MobWS. The element `<startImmediately ...>` corresponds to the immediate invocation of the service. Within this request, 84 bytes of data was transmitted by embedding it in the `<ContextData ...>` element.

With only 84 bytes of input data, the HTTP payload with SOAP interface reaches 1559 bytes. It can be easily observed that the degraded performance of a MobWS provisioning node is directly proportional to the size of input data embedded within the `<ContextData ...>` element. Here, we do not study the influence of increased payload on processing performance, but only compare the message payloads with the REST architecture for similar requests.

Figure 11 depicts the REST message payload transported over HTTP to initiate the same asynchronous `<aEchoString ...>` MobWS creation process. With the REST interface, the service, interaction type, resource and operation is directly idientified by the URL that conforms to the URL structure illustrated in Figure 1(b). With the same amount of input data embedded in the `<ContextData ...>` element (84 bytes), the total payload size is significantly reduced to 356 bytes compared to SOAP. We are able to achieve significant reduction in the response payload as well which was 937 bytes with SOAP, and 105 bytes with the REST architecture.

Similar reduction is HTTP payload has been observed with several types of asynchronous interactions including control and monitor messages. It is expected that the payload reduction will greatly improve the processing and memory performance

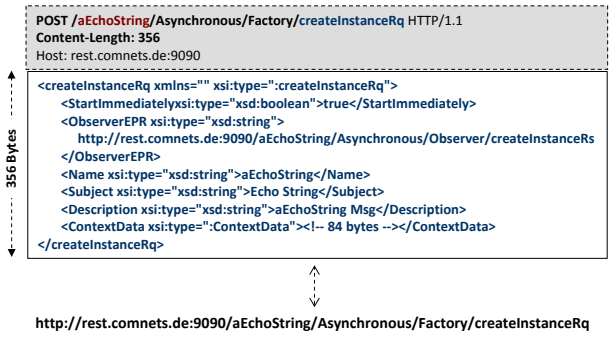



Fig. 11. Asynchronous REST Service Creation Request for aEchoString MobWS

of the MobWS provisioning node. The performance evaluations in terms of processing latencies and are still under research process and a publication is expected soon. However, Figure 12 depicts an overall comparison of HTTP payloads requirements for SOAP and REST architectures for asynchronous MobWS.

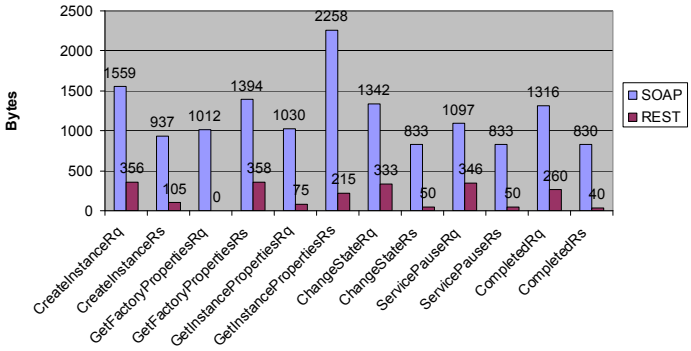



Fig. 12. SOAP and REST HTTP Payloads for Asynchronous MobWS

In Figure 12, a special case can be observed for the operation `GetFactoryPropertiesRq` where no HTTP payload is required for the monitoring process and a 100 % reduction is achieved. This is the only READ request in case of asynchronous interaction, therefore the REST URL is mapped to the HTTP GET method. All other asynchronous messages that we have developed either takes the payload as input, or changes the behavior of the system. Thus, they are mapped to the HTTP POST. In this paper, we do not present all types of asynchronous messages present in [5], therefore more cases like `GetFactoryPropertiesRq` may also exist.

## III. Conclusion

In this paper, a MobWS provisioning system based on the REST architectural principles is presented. In the first part, the concepts of RESTful MobWS and their requirements are discussed in detail in relation to the architectures based on SOAP. Two categories of MobWS interactions, synchronous and asynchronous are identified and the URL structure for each is proposed to expose MobWS using REST interface. The mapping of REST URL to the HTTP methods is elaborated based on the established grounds of architectural requirements. Secondly, the SOAP and REST message structures are presented and compared in terms of HTTP payload requirements. It has been shown that with the REST MobWS architecture, a synchronous MobWS invocation is possible with $\approx 96\%$ reduced payload, whereas $\approx 75\%$ for the asynchronous MobWS when compared to SOAP. A special case is also identified for REST based asynchronous monitoring where 100 % payload reduction is achieved. The MobWS provisioning middleware offers a promising platform to the developers by providing an API supporting efficient and multi-interfaced MobWS. The performance evaluation and comparison of SOAP and REST based architectures in terms of processing and memory requirements is currently under research process and a publication is planned.

## References


[1] F. Aijaz, B. Hameed, and B. Walke. Asynchronous mobile web services: Concept and architecture. In *Proceedings of 2008 IEEE 8th International Conference on Computer and Information Technology*, page 6, Sydney, Australia, Jul 2008. Internet.

[2] T. Berners-Lee, L. Masinter, and M. McCahill. RFC 1738: Uniform resource locators (URL). Published: www.rfc-editor.org, December 1994.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol 1.1. Internet, June 1999. Status: Standard.

[4] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[5] John Fuller, Mayilraj Krishnan, Keith Swenson, and Jeffrey Ricker. Oasis asynchronous service access protocol (asap), 2005.

[6] Guido Gehlen. *Mobile Web Services - Concepts, Prototype, and Traffic Performance Analysis*. PhD thesis, RWTH Aachen University, Lehrstuhl fr Kommunikationsnetze, Aachen, Germany, Oct 2007.

[7] Guido Gehlen, Fahad Aijaz, and Bernhard Walke. An enhanced udp soap-binding for a mobile web service based middleware. In *Proceedings of IST Mobile Summit 06*, page 8, Myconos, Greece, Jun 2006. ComNets, Faculty 6, RWTH Aachen University, Germany.

[8] Guido Gehlen, Fahad Aijaz, and Bernhard Walke. Mobile web service communication over udp. In *Proceedings of the 64th IEEE Vehicular Technology Conference*, page 1, Montral, Canada, Sep 2006. IEEE, IEEE.

[9] Guido Gehlen and Ralf Bergs. Performance of mobile Web Service Access using the Wireless Application Protocol (WAP). In *Proceedings of World Wireless Congress 2004*, pages 427–432, San Francisco, USA, 05 2004. University Aachen, Communication Networks.

[10] Guido Gehlen and Linh Pham. Realization and Performance Analysis of a SOAP Server for Mobile Devices. In *Proceedings of the 11th European Wireless Confernce 2005*, volume 2, pages 791–797, Nicosia, Cyprus, Apr 2005. VDE Verlag.

[11] Guido Gehlen and Bernhard Walke. Transport layer delay analysis in wireless networks using signal flow graphs. In *Proceedings of European Wireless 2007*, page 7, Paris, France, Apr 2007.

[12] Martin Gudgin, Marc Hadley, and Tony Rogers. Web Services Addressing 1.0 - Core. Published on the internet, May 2006. W3C Recommendation.

[13] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter Brown, and Rebekah Metz. Reference Model for Service Oriented Architectures. Published on the internet, December 2005. OASIS Working Draft.

[14] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 805–814, New York, NY, USA, 2008. ACM.

[15] Toshiro Takase, Satoshi Makino, Shinya Kawanaka, Ken Ueno, Christopher Ferris, and Arthur Ryman. Definition languages for restful web services: Wadl vs. wsdl 2.0. Technical report, Tokyo Research Laboratory, IBM Research, 2008.