The Resource-Oriented Mobile Web Server for Long-Lived Services

Fahad Aijaz, Syed Zahid Ali, Muzzamil Aziz Chaudhary, Bernhard Walke
Communication Networks Research Group, RWTH Aachen University
Faculty 6, Kopernikusstr. 16, 52074 Aachen, Germany
E-mail: {fah, muz}@comnets.rwth-aachen.de

***Abstract*—The Web Servers are globally accepted as the backbone of the Internet. The researchers in the area of the enterprise computing have studied, analyzed and implemented several architectures to optimize the performance of the large-scale Web Servers. However, due to the convergence of the IT and the mobile communication domains, the mobile nodes now are also been viewed as the service hosting platforms.**

The aim of this article is to present an optimized Mobile Web Server architecture for provisioning the Mobile Web Services (MobWS) as lengthy processes. For that reason, the asynchronous interaction strategy is derived, and later used to comprehensively discuss the server architecture, which is based on the Representational State Transfer (REST) design principles. The manuscript presents a detailed performance evaluation of the architecture and compares the results with the server based on the Simple Object Access Protocol (SOAP) standard. The results show promising performance improvements due to the reduced payload requirements of the REST server.

I. Introduction

Service Oriented Architecture (SOA) [1] is a globally accepted standard reference model for service-oriented computing, with Web Services (WS) being its most common implementation technology. Today, the Internet and the mobile networks have started to merge based on WS spotlighting high-valued consumer and business use cases. The combination of the mobile communication systems and the WS foresee high economical potential in terms of service provisioning from the mobile nodes, which are known as the MobWS. From the research and technological perspective, the maturity process of the MobWS is ongoing, however, several challenges must be addressed due to the rapid increase in the WS standards that brings new performance challenges for the mobile nodes in terms of thick message payloads. On the other hand, the REST, also termed as Resource Oriented Architecture (ROA), is a software architecture style for distributed hypermedia systems such as the World Wide Web (WWW) [2]. A REST system supports direct interaction with the WS based on the URL [3] standard, which facilitates in avoiding the performance degrading factors such as a bulky SOAP message manipulation. Systems based on REST are tightly coupled with the well known HTTP standard [4]. The decision to choose between the SOAP and REST implementation techniques is based on the use case requirements, however, the research community has written several publications to facilitate the decision process for the enterprise and the mobile computing domains [5], [6].

In continuation of our previous research in [7], the focus of this paper is to study the architecture of the Mobile Web Server that enables *long-lived* MobWS provisioning using the REST design principles. In the first phase, asynchronous MobWS interaction strategy is derived based on the operations specified in the Web Service Description Language (WSDL) standard [8]. Later, the asynchronous server-side architecture and its components are explained in detail by taking into account the REST messaging interface. The final phase evaluates the influence of REST messaging on the architectural performance and compares the results with the similar server that relies upon the SOAP messaging standard. During the evaluation phase, the mean value analysis is also conducted on the collected measurements that shows significantly optimized performance with REST.

II. Related Work

The global acceptance of the MobWS as a mature platform is still in its infancy. Enabling the true potential of the MobWS in the current highly competitive global market faces several research challenges and requires recommendations for efficient solutions at various technical levels. Riva in [9] presents a detailed study, based on the experience of several research projects, about the challenges for developing middleware on smart phones. The importance of resource management, lightweight communication protocols and asynchronous programming are also highlighted. The work in [10], on the other hand, develops a Mobile Host platform for provisioning the MobWS. In this work, in order to reduce the processing latencies within the Mobile Host, BinXML compression technique is adapted. The research in [11] presents the concept of the first ever Mobile Web Server and comprehensively analyzes the traffic performance characteristics of the MobWS. It further classifies the 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 [11], 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 the mobile devices [12]. It is further described by the same group in [13], the technical approach of porting the Apache httpd to the Symbian/S60 mobile platform in order to enable Website hosting and access on the mobile phones.

III. Long-Lived Mobile Web Services

In the traditional consumer-oriented scenarios, the *short-lived* MobWS are widely used to fulfill most of the application requirements. However, in principle, the deployment of the MobWS is not only limited to the consumer focused mobile nodes, such as a smart phone or a PDA, since it depends upon their target application use case. For instance, the services may also be used by a network terminal to perform continuous backend computations, which may help in monitoring a particular environment (eg. Health Care, Surveillance etc.) or to optimize the network/system performance [14]. In such usage scenarios, the short-lived MobWS cannot be directly applied as their instantaneous execution style and blocking communication model (request-response) causes unnecessary resource allocation at the client side [7], [15].

On the other hand, these overheads are avoided by designing a nonblocking communication architecture for the *long-lived* MobWS. The architecture uses the standardized WSDL transmission primitives to derive multiple asynchronous interaction strategies. Eventually, that allows the services to execute asynchronously while the client node remains unblocked. Such long-lived services are termed as 'Asynchronous MobWS'. The following section discusses the asynchronous interaction between the mobile peers that builds the fundamental principals to architect the Mobile Web Server, which is capable of provisioning the asynchronous MobWS.

A. Asynchronous Interaction

Most standard Internet protocols, such as HTTP, are designed with an unspecified presumption of instantaneous feedback that are suitable for short-lived execution model. However, by using the existing standards, an asynchronous interaction style can be derived to support systems that employ WS to perform intricate processing that may take longer time to complete. Designing such an interaction is not straightforward. The standard bodies in WS domain have not explicitly specified asynchronous operations within the standards, however, the mechanisms provided in the current specifications form a fundamental platform for realizing systems based on asynchronous interaction. Christensen et al. in [8] specifies four operations (transmission primitives) namely, one-way, request-response, solicit-response and notification. The one-way (request) and notification (response) operations focus on systems where message delivery guarantees are not required. The request and response with these operations are dispatched as two separate datagram transmissions, enabling high level of decoupling between the service consumer and provider. On the contrary, request-response and solicit-response operation supports message delivery guarantee by relying upon the standard Internet protocol such as HTTP. The request-response operation is similar to the synchronous interaction [7] where each request is followed by a response, whereas reversing the roles of service consumer and provider in this case would lead to the solicit-response operation.

The asynchronous interaction may be derived by incorporating the request-response and solicit-response WSDL operations, however the one-way and notification operations are also possible if delivery guarantees can be compromised. The Figure 1 depicts an asynchronous interaction between P1 and P2 using the request-response and solicit-response operations, where the P2 provisions asynchronous MobWS. The phase of interaction from 1-4 illustrates the request-response operation. When the service invocation request from the mobile application A1 arrives at A2, the service P2-2 is invoked. Attention has to be paid on the generation of response which in this case does not carry the outcome of the invoked service. The asynchronous service starts to function in a separate execution thread, whereas the response dispatched from A2 via the local proxy delivers an acknowledgment to A1 about the successful invocation process. During the entire request-response operation, A1 remains in a blocked state until the acknowledgment arrives. Since the service invocation is a short-lived process, therefore it can be modeled using the blocking behavior at the requester. On the other hand, the invoked MobWS performs long-lived functions independently and must not be a part of invocation process. Once the acknowledgment has arrived, A1 transitions to the unblocked state that allows P1 to perform other necessary functions independently.

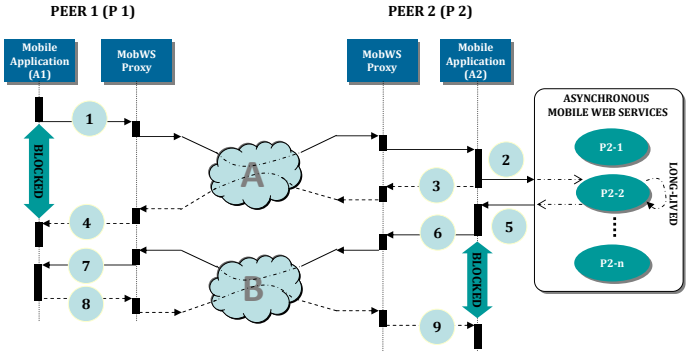

Fig. 1. Reliable Asynchronous Interaction

During the request-response operations, A1 provides an Endpoint Reference (EPR) to A2 that is used to derive the solicit-response operation. The EPR serves as a contact point of A1 that is used to deliver the service outcome or to notify about alerts and exceptions. The requester during the service invocation process may provide an EPRof a different mobile or fixed terminal, if the responses from A2 are required to be delivered to a different network node. Assuming in the current scenario, the response from A2 is delivered to the requesting node, the second phase from 5-8 illustrates the solicit-response operation. Upon completing the processing, P2-2 forwards the outcome to A2 that delivers it through the local proxy to A1. The local proxy at P1 receives and forwards the result to A1, which acknowledges A2 by sending a response message. Until the acknowledgment has arrived, the corresponding thread of A2 remains in a blocked state.

Both the operations rely upon the synchronous interaction with per-request acknowledgment behavior, however, combin-

ing both in a single system communication strategy leads to a reliable asynchronous interaction. The one-way and notification operations of WSDL supports the unreliable asynchronous interaction that can be easily achieved by avoiding the acknowledgments for each transmission primitive respectively. The details regarding the unreliable interaction are, therefore, not presented here. In principle, the interaction is independent of the underlying network infrastructures. The two network clouds **A** and **B** in the figure depicts this behavior where P1 and P2 communicate over the distinct networks using the request-response and solicit-response operations.

IV. REST-Interfaced Asynchronous Mobile Web Server

The design of the Mobile Web Server architecture using REST principles is not straightforward. The implementation of MobWS using the traditional and mature SOA reference model is highly flexible and extensible toward the heterogeneous networks and changes due to the transport neutral behavior of SOAP. The variety of existing WS standard specifications adds to the global acceptance of SOA for enterprise systems. This however leads to coarse-grained and thick messaging structures, especially in case of mobile terminals.

On the other hand, 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 standards can be optimized to be employed by REST architectures in order to handle the requirement changes. By definition, in the REST-interfaced MobWS access, the HTTP is not only used as a transport protocol, but also conveys the actions that must be performed at the serving node. This demands the REST architecture to establish the mapping of requested URL with HTTP methods that defines the purpose of the clients' request. On the other hand, the service provisioning node, upon receiving the request, must be able to understand the mapping in order to take intended actions. The REST-based asynchronous MobWS access using the aforementioned URL and their mapping to HTTP methods is already presented in a recent publication. Therefore, the focus of this paper is to present the architectural and performance aspects of the REST-interfaced asynchronous Mobile Web Server. For further insight into area of REST-based asynchronous MobWS messaging, the study of [15] and [7] is highly recommended.

A. Server Architecture

The system proposed in [11] uses SOAP Remote Procedure Call (RPC) mechanism to consume MobWS. Thus, every SOAP request uses *almost* the same URL structure as discussed in [15]. The difference is created by replacing the SERVICE parameter 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 the POST HTTP method is used.

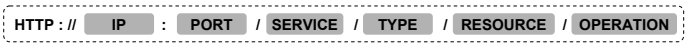

Fig. 2. The Asynchronous REST URL Structure

On the other hand, any request for consuming MobWS over the REST interface conforms to the URL structure shown in Figure 2 (refer to [15]). Therefore, for the REST requests, the parameter *soaprpc* is not required which eases the request type identification process by relying upon the structure of the received URL. Hence, the request listening component of the server identifies the target architecture of the received request by checking for the *soaprpc* parameter in the HTTP URL.

The further discussion of the service-side components is classified into the following three internal processes of the server.

1) The Service Creation Process: Figure 3 illustrates the asynchronous service invocation process using the REST server architecture in a mobile terminal. Here, the flow of information and control transitions between different server components is shown. Upon the arrival of client's request REQ, the HTTP Listener (L) component identifies the request type based on the soaprpc parameter, as explained previously. Assuming the request has arrived for the REST interface, L initiates a thread called Request Processor (ReqP) by providing REQ and request type identification flag, which is depicted in the figure as REST. The ReqP is a parent server component responsible for managing incoming requests in accordance to their requirements. It is also responsible for delegating and initializing other server components on demand. Since the REQ targets the MobWS over the REST interface, 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 to conform with the REST design principles. The responsibility of RM is to extract all the information embedded within the 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 the MobWS. 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 the HTTP payload and check for the 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. Then, the REST Container object (RCo) is returned to the ReqP by RM once it is created. Since the information delivered by the client in REQ is now encapsulated inside 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 the RM, the ReqP delegates the control flow, along with the RCo, to the Asynchronous Manager (AM) which is gateway to enter the asynchronous

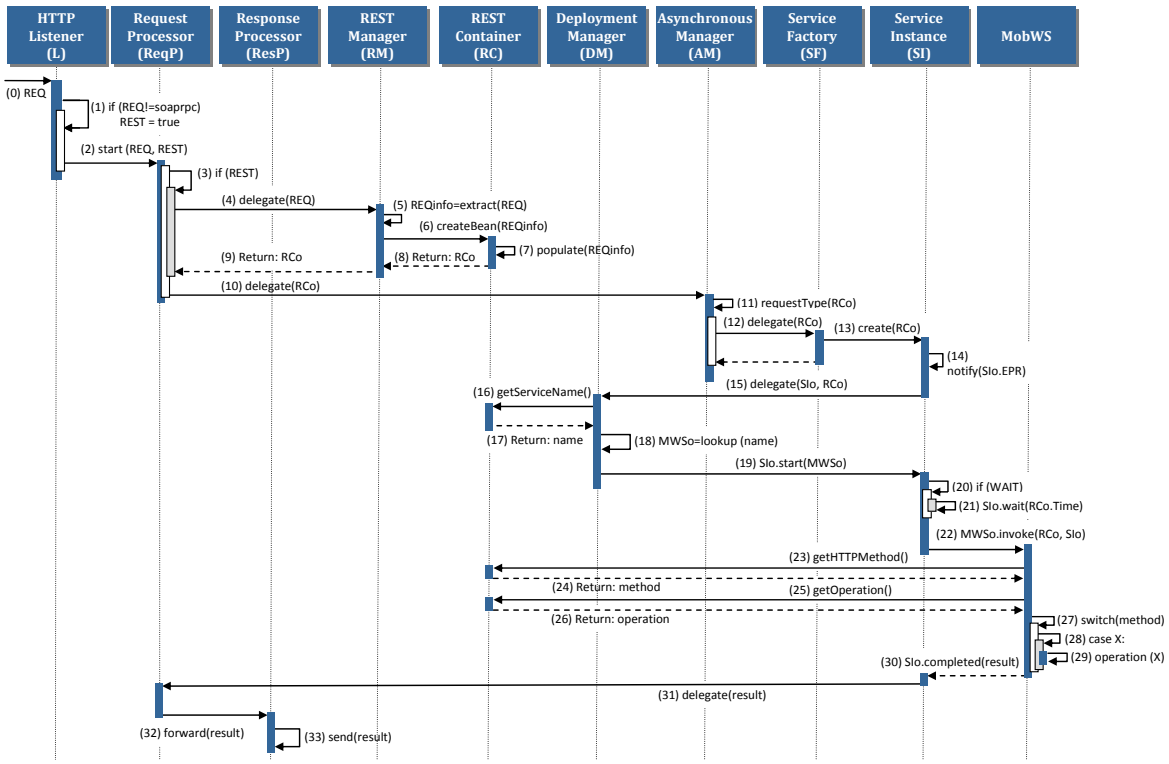

Fig. 3. Sequence Diagram of the Server-side Architectural Components

server architecture. Due to the conformance of the architecture to the asynchronous service access standard [16], the server is capable of exposing variety of resource methods that a client may invoke (refer to [17], [18]). Assuming the client targets an immediate MobWS invocation, the AM identifies the type of request and delegates the control to the Service Factory (SF). The SF is a singleton component, whose one of the major roles is to create the Service Instance (SI) in case of the MobWS invocation request. As soon as the SI is created, it notifies the requesting client about its successful creation. At this phase, the request-response operation of the asynchronous interaction is completed that releases the client from the blocked state (see Section III-A). With the notification message, the SI also provides its unique EPR that the client may use for the future communication. This is to note that the SI is *not* a MobWS, rather, it exposes an interface to the outside world that may be used to perform several kinds of service management functions that are specified in the standard [16]. Thus, for every instance of the target asynchronous MobWS, a wrapper of SI is created which is uniquely identified via the its unique EPR. After the notification dispatch, the SI delegates the control to the Deployment Manager (DM) along with the RCo and its own object, SIo. The DM maintains a list of MobWS that are deployed on the server along with their corresponding objects in form a key-value map. Thus, it uses the RCo to fetch the requested service name and then uses it to lookup the corresponding MobWS object (MWSo). Subsequent to that, the SIo is used by the DM to start the SI thread. The MWSo is also forwarded to the thread that is eventually required for the service invocation.

2) The Service Invocation Process: The invocation process of the asynchronous MobWS may be categorized as *instantaneous* or *scheduled*. The instantaneous process is straightforward in which the service is invoked as soon as the MWSo is received by the SI. On the other hand, in the scheduled invocation process, the requesting client specifies the exact time in future when the MobWS must be invoked by the SI. When the MWSo is received by the SI thread, it uses the RCo to identify the type of invocation process intended by the client. Thus, in case of the scheduled invocation, the SI transitions to the waiting state until the specified time in the RCo is expired. Eventually, irrespective of the invocation process, the target MobWS thread is invoked by the SI using the MWSo. To facilitate the two-way communication between the SI and the service, and to ensure the execution of the correct service operation, the SIo and RCo are also delivered to the MobWS when it is invoked.

3) The Service Execution Process: Generally, the MobWS execution process defines the functions that a service may offer to its clients. For this reason, the process is completely dependent upon the service requirements and how it is implemented by the developer. However, the MobWS deployment architecture of the server describes few fundamental steps that the service developer may follow. Here, one possible example of the MobWS execution is provided, but many other approaches to implement the service do exist.

Once the control flow is delegated by the SI to the target

MobWS, the process of identifying the target service operation must be performed. This is done by first obtaining the mapped HTTP method and the targeted operation using the RCo, and then establishing the correlation between the two, which identifies the correct operation that the service shall execute. Consider a situation in which a 'Location' MobWS offers multiple operations, such as, Elevation, Address, Coordinates etc. In this case, the correlation between the URL and the HTTP method entirely changes the context of a request. For example, in order to *fetch* the Elevation, the requester may map the URL to the HTTP GET method, which clearly indicates the targeted operation, getElevation(). Similarly, to *update*, *insert* or *delete* the Elevation, the same URL may be used with different mappings to POST, PUT and DELETE methods respectively. Eventually, upon the completion of the target operation, the service provides the final result to the SI that delegates it to the ReqP. The ReqP forwards the result to Response Processor (ResP) component which sends the result to the client.

V. Performance Evaluation

The performance evaluation of the aforementioned architecture is conducted to study the processing latencies during the Service Creation Process (SCP). The experiments are carried out in an emulated environment in which the SCP of an *echoString* MobWS is executed several times over the REST and SOAP messaging interfaces with a constant payload. The evaluation is limited to the SCP, because it defines a mandatory prerequisite for any asynchronous MobWS to function while being transparent towards the service messaging interfaces. However, some architectural components may differ that we will discuss if and when necessary.

A. Evaluation with SOAP Messaging

The architecture of the Mobile Web Server slightly differs in terms of the components when the SOAP messaging interface is used. This is because the messaging relies completely upon the pre-standardized messaging framework [19] to execute the target functions, and the URL is only used to identify the client and the server nodes. Furthermore, the SOAP messaging interface requires additional WS standards [20] to address session management issues. For these reasons, the server-side components are designed to manipulate and understand the incoming requests, so that the encapsulated information within these standards can be used to execute the target server functions, such as the SCP.

In the SOAP architecture, the *REST Manager (RM)* and *REST Container (RC)* components are not used as they are designed only to meet the requirements of the REST messaging interface. All the remaining components are capable of handling both types of requests. In the Figure 4, the collected server processing latencies for the SCP over SOAP interface are presented. In the first phase, the experiment was conducted for 50 sequentially executed requests. The measurements shows that the mean latency of the asynchronous SOAP server after 50 request is $76.62ms$. Relative to the mean value, the calculation of the positive and negative standard deviations implied the maximum deviated latency of $87.33ms$, whereas the minimum deviation resulted in $65.91ms$.

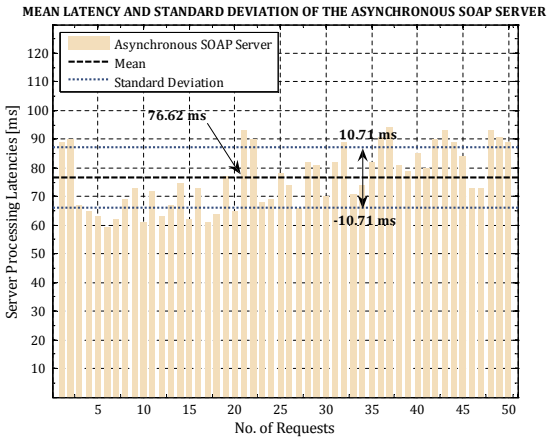

Fig. 4. SOAP Server During the Service Creation Process

In the second phase of the experiment, the individual latencies of the server components were separately collected and compared, as shown in the Figure 5. The mean value analysis showed that the component *Response Processor (ResP)* alone caused $\approx 66.6\%$ of the overall server lantency. The further study pointed out that due to the dependancy of SOAP on the Extensible Markup Language (XML), the construction and the parsing of the response messages for the client results in significant processing delays at the server. On the other hand, the *Request Processor (ReqP)* component added $\approx 28\%$ to the total server delay with its mean individual latency of $21.24ms$. The processing delay of the ReqP was less than the ResP because it does not carry out the functions of constructing the SOAP message. For this reason, most of the latency in ReqP is caused by the SOAP messaging parsing of the incoming request, and the error checking functions, such as the validity of the XML data. Other than that, the mean latency of the *Asynchronous Manager (AM)* was negligible and only added $\approx 0.4\%$ to the server processing. The latencies of the other components involved in the SCP, such as the *SF*, the *DM* and the *SI* were significantly negligible, and therefore, are not shown in the results.

For the details regarding the asynchronous SOAP messaging and architecture, the reader is referred to [15] and [17].

B. Evaluation with REST Messaging

Unlike in the SOAP messaging, the REST access interface strictly couples with the URL and the HTTP methods to convey the targeted functions. For this reason, the dependency of REST on XML based message structures is significantly reduced that leads to the simplified request handling at the server. In this phase of the evaluation, the architecture discussed in Section IV-A1 was used to conduct the same experiment with the identical testbed environment as presented in the SOAP scenario. The experiment was started by sequentially dispatching 50 requests to the Mobile Web Server in order

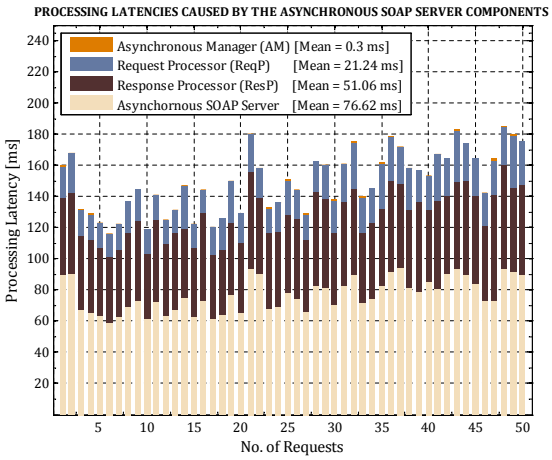

Fig. 5. Latencies Caused by the Individual SOAP Server Components

to perform the SCP using the REST messaging interface. The collected measurements showed promising improvements, which are shown in the Figure 6. The overall mean server processing latency with asynchronous REST server resulted in $15.8ms$, with the relative maximum and minimum deviations of $22.293ms$ and $9.307ms$ respectively.

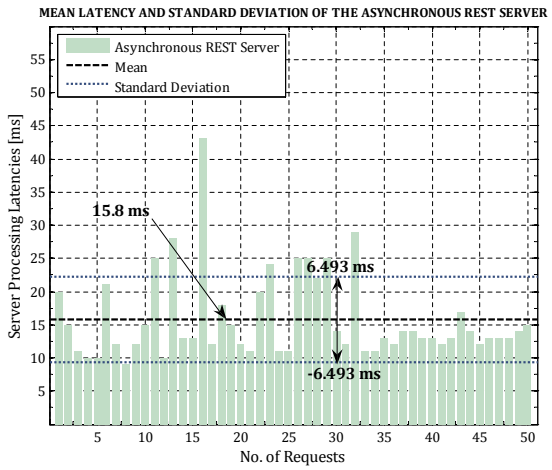

Fig. 6. REST Server During the Service Creation Process

Similar to the previous experiment, the second phase of the evaluation focused on analyzing the individual server components of the REST architecture. The analysis of the mean values showed that the ReqP component in case of REST caused $\approx 37\%$ of the overall server delay. This is becaused the targeted server functions were completed identified using the request URL, thus, the XML manipulation in the component was avoided. In addition to that, the mean latency of the RM component resulted in $5.84ms$, which is $\approx 36.9\%$ of the total server latency. The mean latency of the RM presented here also includes the delay caused by the RC component. Since in the REST architecture, the RM is a child process of the ReqP, therefore the *real* latency of the ReqP can be identified by deducting the latency of the RM from that of ReqP. Therefore, the outcome leads to only $0.06ms$ delay that is actually caused by the ReqP alone. This calculates to $\approx 0.3\%$ of the overall server processing latency. Moreover, the ResP took $4.34ms$ to generate and dispatch the response to the client, which add $\approx 27\%$ to the overall delay. Unlike in SOAP, the ResP in REST only generates a new response URL, and the service result is directly embedded in the HTTP payload. Consequently, the processing demands, such as the XML construction and parsing, are omitted and the processing delay is significantly reduced. Other than that, the AM added a negligible value of $0.44ms$ to the overall server processing, whereas, the *SF*, the *DM* and the *SI* also resulted in the negligible delays which are not presented here.

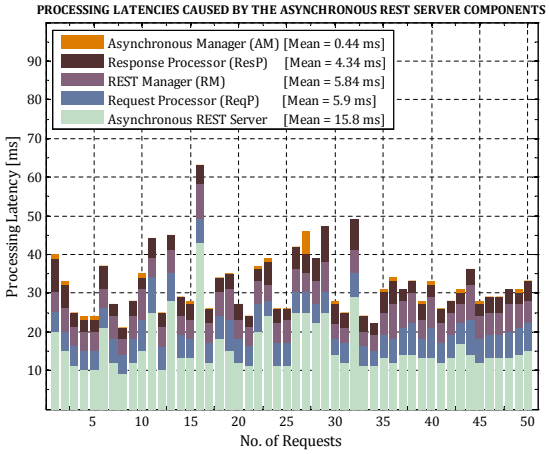

Fig. 7. Latencies Caused by the Individual REST Server Components

VI. Conclusion

The research presented in this paper focused on the architecture of a Mobile Web Server to enable the provisioning of WS with lengthy execution demands. In that context, the work tries to bring the REST design principles into a mobile setting by presenting an asynchronous REST server architecture for the mobile nodes. Moreover, the manuscript discusses in detail the aspects of asynchronous interaction which are derived from the WSDL standard. Later, the interaction is used to provide the in-depth explanation of the server architecture and its components. The role of each component is clearly presented by classifying the server functions into the internal server processes, namely, the *Service Creation Process*, the *Service Invocation Process* and the *Service Execution Process*.

The performance evaluations of the asynchronous REST Mobile Web Server showed promising optimizations in terms of processing delays at the server. The overall results based on these experiments showed that the REST server is able to process the SCP $\approx 79\%$ faster than the SOAP server. The evaluation of the server-side components spotlighted that the overall mean performance of the REST architecture is significantly less than the individual latencies caused by the most SOAP server components. The comparison of the component-level latencies in both the server architectures also shows

promising optimization with the REST messaging interface.

To conclude, the REST architecture for the Mobile Web Servers addresses the issues related to the high-payload demands of the MobWS, and might be useful to explore new design approaches for mobile applications of the future Internet.

REFERENCES

[1] C. M. MacKenzie, K. Laskey, F. McCabe, P. Brown, and R. Metz, "Reference Model for Service Oriented Architectures," Published on the internet, Dec. 2005, oASIS Working Draft. [Online]. Available: http://www.oasis-open.org

[2] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

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

[4] 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. [Online]. Available: ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt

[5] C. Pautasso, O. Zimmermann, and F. 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*. New York, NY, USA: ACM, 2008, pp. 805–814.

[6] T. Takase, S. Makino, S. Kawanaka, K. Ueno, C. Ferris, and A. Ryman, "Definition languages for restful web services: Wadl vs. wsdl 2.0," Tokyo Research Laboratory, IBM Research, Tech. Rep., 2008. [Online]. Available: http://www.ibm.com/developerworks/library/specification/ws-wadlwsdl/index.html

[7] F. Aijaz, S. Z. Ali, M. A. Chaudhary, and B. Walke, "Enabling resource-oriented mobile web server for short-lived services," in *Proceedings of the Ninth IEEE Malaysia International Conference on Communications (MICC 2009)*. Kuala Lumpur, Malaysia: IEEE, December 2009, p. 6.

[8] 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

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

[10] 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/

[11] 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

[12] J. Wikman and F. Dosa, "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

[13] J. Wikman, F. Dosa, and M. Tarkiainen, "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

[14] F. Aijaz, S. M. Adeli, and B. Walke, "A service-oriented approach for in-network computations in wireless networks," in *In Proceedings of the Sixth IEEE and IFIP International Conference on Wireless and Optical Communications Networks*. Cairo, Egypt: IEEE / IFIP, April 2009, p. 6.

[15] F. Aijaz, S. Z. Ali, M. A. Chaudhary, and B. Walke, "Enabling high performance mobile web services provisioning," in *In Proceedings of the 2009 IEEE 70th Vehicular Technology Conference (VTC2009-Fall)*. Anchorage, Alaska USA: IEEE, September 2009, p. 6.

[16] J. Fuller, M. Krishnan, K. Swenson, and J. Ricker, "Oasis asynchronous service access protocol (asap)," 2005. [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=asap

[17] 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*. Sydney, Australia: Internet, Jul 2008, p. 6. [Online]. Available: http://www.comnets.rwth-aachen.de

[18] F. Aijaz, H. Bilal, and W. Bernhard, "Towards peer-to-peer long lived mobile web services," in *Proceedings of the 4th International Conference on Innovations in Information Technology*. Dubai, UAE: IEEE, Nov 2007, p. 5. [Online]. Available: http://www.comnets.rwth-aachen.de

[19] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework," Published on the internet, Jun. 2003, w3C Recommendation. [Online]. Available: http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[20] M. Gudgin, M. Hadley, and T. Rogers, "Web Services Addressing 1.0 - Core," Published on the internet, May 2006, w3C Recommendation. [Online]. Available: http://www.w3.org/TR/ws-addr-core/