Implementation and Evaluation of a JSON Binding for Mobile Web Services with IMS Integration Support

SHAHID MURTAZA

Master's Degree Project
Stockholm, Sweden

XR-EE-LCN 2012:002
Master Thesis

Implementation and Evaluation of a JSON Binding for Mobile Web Services with IMS Integration Support

Implementierung und Evaluation eines JSON Binding für Mobile Web Services mit IMS-Integration Support

Of

Shahid Murtaza

Matriculation Number: 299656

Aachen, December 5, 2011

Supervised by:
Univ. Prof. Dr.-Ing. B. Walke
Fahad Aijaz, MSc

This publication is meant for internal use only. All rights reserved. No liabilities with respect to its content are accepted. No part of it may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.
I assure, that this work has been done solely by me without any further help from others except for the official attendance by the Chair of Communication Networks. The literature used is listed completely in the bibliography.

Aachen, December 5, 2011

(Shahid Murtaza)
Service Oriented Architecture SOA is a well recognized and immensely applied reference model for service oriented computing, both in industry and research. Because of its fastidious features, such as reusability, interoperability, scalability and flexibility a Web Based SOA is adopted, in the areas like Enterprise Application Integration EAI. These days the Web Based Services, such as XML Web Services are not limited to fixed servers but can also be deployed on mobile devices in order to enhance the capability to become the service provider as well, such Web Services are known as Mobile Web Services MobWS. With the limited capabilities of mobile devices, the performance of XML based Mobile Web Services is affected due to the requirements of XML manipulation and parsing over the Representational State Transfer REST architecture.

This Thesis work aims to develop and design a lightweight JSON Binding using the REST architecture. This will enhance the REST architecture to enable service creation and management for the synchronous as well as for the asynchronous Mobile Web Services MobWS. Initially, the existing synchronous and asynchronous server architecture is analysed to establish the foundation of the JSON Framework. Then the JSON Binding is developed in conformance with the analysed architecture. Therefore, the existing XML messaging constructs are redefined and optimized by using the JSON. Additionally JSON Binding developed in this work is used to enable service consumption in the IMS. This work is done by developing IMS client side modules in the middleware using Ericsson’s Mobile Java Communication Framework (MJCF). Finally the performance analyses have been done to evaluate and compare the impact of JSON Vs XML Binding over the synchronous as well as for the asynchronous server architecture. Thus, the architectural capability of the MobWS middleware has been customized to select the appropriate content type to handle and process the client requests.
1. Introduction ................................................................................................................. 9
  1.1 Motivation ............................................................................................................... 9
  1.2 Thesis Scope ........................................................................................................... 9
  1.3 Thesis Structure ...................................................................................................... 9
2. Background .................................................................................................................. 11
  2.1 Web Services ......................................................................................................... 11
    2.1.1 Mobile Web Services ......................................................................................... 11
  2.2 Introduction of JSON .............................................................................................. 11
    2.2.1 JSON Data Structure ....................................................................................... 12
    2.2.2 JSON Specification ......................................................................................... 12
3. JSON Binding over REST Framework ......................................................................... 14
  3.1 Existing Middleware .............................................................................................. 14
    3.1.1 Messaging Format ............................................................................................. 15
    3.1.2 Overhead of XML Format ............................................................................... 15
  3.2 JSON Vs XML Messages over REST Framework .................................................... 15
    3.2.1 Synchronous Operations ............................................................................... 16
    3.2.2 Asynchronous Operations .............................................................................. 17
  3.3 Activity phases of the Mobile Server ..................................................................... 20
    3.3.1 Request Receiving Phase ............................................................................... 20
    3.3.2 Request Handling Phase ............................................................................... 21
    3.3.3 Response Handling Phase ............................................................................. 30
4. Performance Evaluation of Mobile Web Services ..................................................... 32
  4.1 Overview ............................................................................................................... 32
  4.2 Performance Evaluation Criteria ........................................................................... 32
    4.2.1 Testbed ......................................................................................................... 32
Chapter 1

Introduction

1.1 Motivation
Because of the increasingly distinct capabilities of Web Services clients, today Web 2.0 supports Web Services with multiple content types to address diverse types of clients. Multiple content types is also the fundamental requirement of IP Multimedia Subsystem (IMS) as it is all about enriching communication among a variety of consumers. However in the existing Mobile Web Services MobWS Framework there is no support for multiple content types, which narrows the scope of hosting Web 2.0 services on the mobile devices. On the other hand XML manipulation and parsing on mobile nodes is computationally expensive process, due to high communication cost and unexpected connectivity. Therefore, the performance of the Mobile Web Services MobWS Framework is degraded by adopting such insufficient techniques. The introduction of new content type may also optimize the performance factor.

1.2 Thesis Scope
This Thesis will introduce and develop a generic lightweight JSON messaging format and integrate it with the existing REST framework. The work requires the study and analyses of the existing synchronous (short-lived) and asynchronous (long-lived) server architecture to establish the foundation of the JSON Framework. Then the JSON Binding over the REST framework will be develop in conformance with the analysed architecture. Therefore, the existing XML messaging constructs shall be redefine and optimize by using the JSON. The development of multiple content type feature over the MobWS middleware broaden the scope of hosting Web 2.0 services on the mobile devices to address diverse types of mobile clients. This will enhance the REST architecture to enable service creation and management for the synchronous as well as for the asynchronous Mobile Web Services MobWS. The development of JSON messaging format is also used to bring IMS client side compatibility to the MobWS middleware using Ericsson Mobile Java Communication framework (MJCF). At the end, the functionality and conformance of the enhanced framework shall be verified. Finally, to identify and compare the performance of multiple content types (JSON Vs XML) over the REST framework is extensively evaluated by invoking both synchronous and asynchronous MobWS.

1.3 Thesis Structure
Initially the techniques and the concepts of the existing framework are discussed, which are related with the design and development of the JSON messaging constructs. Next the related work is examined which is either being done or is currently going on in the area of Service Oriented Computing by other Research Groups. From this study we have figure out the design principles that will be used for the development work to make our MobWS middleware more advance and compatible. Next after discussing the existing system techniques and the causes of the overhead, the implementation and deployment of the modified framework shall be discussed. It will also include the reasons of deploying multiple content types feature with the emphases of using JSON as a messaging format. To evaluate and compare the multiple content types JSON Vs XML over the REST architecture, a detailed Performance Analyses has been done. Finally the work is concluded by
discussing its strengths and weaknesses. Below you will find a quick overview of the remaining chapters in the Thesis work.

Chapter 2: Background: This chapter gives an overview of the fundamental concepts required to understand the motivation of the thesis work explained earlier and the thesis concepts used in rest of the report. In this context, we discuss the MobWS concepts and the structure of the framework, which is applied into practice during the implementation work for the MobWS middleware.

Chapter 3: JSON Binding over REST Framework: This chapter includes a brief overview of the existing REST architecture of the MobWS middleware. Afterwards, it concisely compares the JSON and XML messaging constructs to give the readers an overview of maximum utilization of the resources of the mobile devices in terms of content length and parsing. After adopting the optimized JSON messaging construct, different phases of the MobWS middleware are explained all the way through the help of the system flows and the messaging constructs. In addition, the reasoning is also provided with the implementation to understand the modified MobWS middleware in a better way.

Chapter 4: Performance Evaluation of Mobile Web Services: This chapter discusses the performance of the JSON enabled Mobile Web Services MobWS middleware, focusing mainly on two aspects i.e. The additional overhead acquired by introducing the JSON framework in the existing systems. And the behaviour of different components in the existing REST based systems, when it works in combination with both the JSON and XML framework.

Chapter 5: Theses: This chapter briefly outlines the thesis’ results.

Chapter 6: Conclusion and Outlook: This chapter summarizes the thesis work and suggests some future extensions and enhancements that can be done to progress further in the area of Mobile Web Services MobWS.
Chapter 2

Background

This chapter gives an overview of the fundamental concepts required to understand the motivation of the thesis work explained earlier and the concepts used in rest of the thesis report. In this context, we discuss the MobWS concepts and the structure of the framework, which is applied into practice during the implementation work for the MobWS middleware.

2.1. Web Services

The term Service is described in [2] as follows:

“Services are autonomous, platform-independent entities that can be described, published, discovered, and loosely coupled in novel ways. They perform functions that range from answering simple requests to executing sophisticated business processes requiring peer-to-peer relationships among multiple layers of service consumers and providers. Any piece of code and any application component deployed on a system can be reused and transformed into a network-available service.”

The Services deployed over the internet are known as Web Services. The development of internet based services provides benefits to the service consumers to easily acquire data and functional services. Many of the well known industry and research institutes are obtaining high interest in the development of Web Services platforms after looking into the increasing growth of the SOC using the Web Services during the recent years.

2.1.1. Mobile Web Services

Service Oriented Architecture SOA using Web Services is a well recognized and immensely applied reference model for service oriented computing, both in industry and research. Because of its fastidious features, such as reusability, interoperability, scalability and flexibility a Web Based SOA is adopted, in the areas like Enterprise Application Integration EAI. These days the Web Based Services, such as XML Web Services are not limited to fixed servers but can also be deployed on mobile devices in order to enhance the capability to become the service provider as well, such Web Services are known as Mobile Web Services MobWS.

Generally mobile devices have limited processing power and lower memory. Therefore, these limitations should be considered in mind prior to the implementation of the MobWS systems, as it effects the manipulation and parsing, which are the important factors to obtain Quality of Service.

2.2. Introduction of JSON

JSON stands for Java Script Object Notation. Recent studies about the web services shows a great number of new emerging web services return very simple data types like number, strings, objects etc, which supports the introduction of a data structure which is based on JavaScript. The fore mentioned data types can be easily serialized as a string, the same format as JSON.
2.2.1. JSON Data Structure

JSON is an emerging messaging format which includes data structures shown in Figure 2.1, like integers, objects, characters, Boolean, or a null value. A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. An object is an unordered collection of name/value pairs. An array is a well-organized set of values.

![JSON data structure](image1.png)

Figure 2.1: JSON data structure

2.2.2. JSON Specification

JSON was specified by Douglas Crockford and is described in RFC 4627. It is a subset of the standard ECMA-262, dedicated to the standardization of Information and Communication Technology (ICT) and Consumer Electronics (CE). JSON has its roots in many programming languages (e.g. C, Perl, PHP and JavaScript), data types and structures and therefore provides an optimized mapping format to exchange structured data. It supports text based format and can be easily parsed by the machines. JSON has a lightweight programming-based format so it can be read more easily by other programming languages. Also JSON has all types of data structures that are present in the XML format.

![JSON Format](image2.png)

![XML Format](image3.png)

Figure 2.2: JSON Format

Figure 2.3: XML Format

In Figure 2.2 describing the JSON messaging format the content length is approximately 28.5 % smaller as compared to the content length shown in Figure 2.3 describing the XML messaging format. The reason for the difference in the content length is due to the content duplication as
shown in Figure 2.3, as 7 extra bytes are required to describe the word “Germany” in case of XML messaging format, which of course results in memory consumption, which is a quite expensive feature while considering the limitations of the mobile devices.
Chapter 3

JSON Binding over REST Framework

REST provides software design principles, through which resources on the internet can be accessed with the help of the URI. The REST style, which we are using, is Client-Server. So URI plays a vital role in the identification of the resources, as the clients can easily access any information on the internet by using a URI, which targets a particular resource. In this chapter we shall discuss the technical information about the existing system, which is based on the REST architecture. In our existing system the messaging format used is XML, which is used between client and server in order to communicate. Than we shall put forward the overheads of using XML as a messaging format. Further we shall introduce a new binding over the REST framework known as JSON and then compare the JSON messaging format with the XML messaging format. In addition, we shall discuss the implementation of the modified framework.

3.1 Existing Middleware

These days, Mobile Web Services MobWS are playing an important role in increasing the potential of the mobile devices. New features in the mobile devices, such as, web browsing, photo sharing, internet radio and navigation, works in parallel with the previously existing services like voice and short messaging. With the development of the Mobile Web Services middleware (MobWS) now it is possible to host as well as consume these Web Services on the mobile nodes. Figure 3.1 demonstrates the MobWS middleware used for the REST architecture.

![Figure 3.1: REST Mobile Web Services Middleware](image)

The existing REST MobWS middleware is capable to serve both HTTP and UDP requests. The server is always in the listening mode in order to accept the requests from the clients. Therefore for this reason, the respective listener i-e HTTP listener in case of HTTP requests or UDP listener in case of UDP requests, receives the incoming request from the client and passes it to the RESTful Handler.
shown in figure 3.1. In case of synchronous request, the RESTful Handler after acquiring the required information sends the control to the Deployment Interface. The Deployment Interface holds the information regarding all the available MobWS, which it acquires through the Service Register interface. In the Deployment Interface, service invocation procedure is performed. Here it waits for the response from the service. When the response is received it will transfer the control to the Response Handler, in order to send the response message back to the client.

In case of asynchronous service request, the control from the Request handler moves to the ASAP Handler shown in figure 3.1. For service creation and invocation in case of synchronous, as well as, asynchronous MobWS, we shall discuss different phases in detail further in this chapter.

3.1.1 Messaging Format
In our existing MobWS middleware we were using XML as a messaging format. The XML format was used in order to construct and represent the request and the response messages between the mobile client and the mobile server.

3.1.2 Overhead of XML Format
The content duplication and the tags used in the representation of XML results in an overhead while using XML as a messaging format. As shown in figure 3.2, the duplication of the word “Germany”, results as an overhead in the memory, due to increase in the number of bytes. In this example we are discussing a message with a very short size of Content Length 172 bytes, however as the message size increases exponentially, the content duplication also increases dramatically, which results in high memory consumption. In the memory the XML elements are hosted like a tree as shown in Figure 3.3. The tree grows as exponentially as the message size grows and in order to parse the XML data by the mobile nodes, the parser traverses whole of the XML tree which is a computationally expensive process.

3.2. JSON Vs XML Messages over REST Framework
In order to give a detailed comparison overview between the JSON messaging format and the XML messaging format, it would be necessary to describe the details regarding the request-response messages for synchronous, as well as, asynchronous MobWS. This overview shall demonstrate the
parsing and memory consumption comparison between these two messaging format in a much better way.

3.2.1. Synchronous Operations

In HTTP synchronous operations, the mobile client uses the HTTP socket in order to communicate with the mobile server. After opening the socket the client sends its request and waits for the response. To get a successful response it keeps the socket open. After receiving a successful response the client closes the socket.

For the synchronous MobWS the request messages from the client consists of HTTP header and body. HTTP header consists of the HTTP request method, service name, Content Type and Content Length. During the client request, for both the JSON and XML binding, the HTTP request method and the service name remains the same. However in case of JSON request message, the Content Type changes to “application/json”, while in case of XML request message the Content Type is “text/xml”, as shown in Figure 3.4. The Content Length in the request message describes the number of bytes contained in the request body. Figure 3.4 shows the Content Length in case of JSON request is approximately 23 % smaller than the Content Length in case of XML request message. The increase in the Content length, in the XML request message, is due to data duplication in XML messaging format. As the Content Length reduces remarkably for large JSON messages, which makes it light weight and of course, reduces the total processing time.

![Figure 3.4: JSON Vs XML over REST Binding](image)

In the synchronous MobWS message, all the communicating nodes, in our case mobile client and server correspond simultaneously, to send and receive the request-response messages between each other. The mobile server attaches a status code of 200 OK in the HTTP header, in order to send a successful response message back to the mobile client. Additionally, the HTTP header comprises of the Content Type and the Content Length. Moreover the response body in the response message consists of the information (service data), which was requested by the mobile client. Figure 3.5 shows the synchronous response message of the “Country Search” request by the mobile client.
3.2.2. Asynchronous Operations

The HTTP asynchronous operations are the enhancement of the synchronous operations. As to fulfill the situations, where the request-response time between the client and server exceeds a lot, like days, week or more. The scenario like these evolved the concept of asynchronous communication, in which the client after sending the request through the internet socket can close the socket. However before closing the socket the client gets a specific parameter from the server, in order to communicate in future. To demonstrate HTTP asynchronous operations in detail, I have discussed two of the asynchronous operations.

3.2.2.1. Resource creation.

3.2.2.2. Resource monitoring.

3.2.2.1. Resource creation

To create an instance, the client sends HTTP POST method with a specified collection of service resources or sub resources. The URI format of the client create instance request is defined in REST URI scheme.

The create instance request is composed of HTTP header and HTTP body. In asynchronous operations like synchronous operation, the Content Type within the HTTP header is “application/json” in case of JSON request. While the Content Type is “text/xml” when a XML request message was sent to the mobile server as shown in Figure 3.6. The Content type “text/xml” demonstrates, the client is sending a text document, in the body of the request. The HTTP body contains all the parameters which will be used to create a new instance. For a successful instance creation, the text document should be a valid representation of XML. The Content Length in the HTTP header defines the message size. Figure 3.6 shows the message size in case of JSON create instance request is approximately 23 % smaller than the message size in case of XML request. The
reason for small message size is the JSON does not apply data duplication while creating a request message.

![Figure 3.6: JSON Vs XML Create Instance Request](image)

As the asynchronous operations are the long duration operations, the mobile client before closing the internet socket requests for the instance EPR from the mobile server. To have the knowledge about the status of the operation, the server sends an instance EPR in response, through which the client can gather the information in future regarding a specific instance. Figure 3.7 shows the create instance response message, which gives a comparison between both the JSON and XML messages.

![Figure 3.7: JSON Vs XML Create Instance Response](image)

3.2.2.2. Resource monitoring

The MobWS instance can be monitored by the client through pause, stops and terminates request during their execution. The state change operation is used to change the state of the predefined instance to a new state.
Such as instance creation request, state change request is also composed of HTTP header and HTTP body. For the state change request, in the HTTP header shown in figure 3.8, the Content Type for the JSON request is “application/json”, while the Content Type for the XML request is “text/xml”. The Content Length required for representing the JSON request is approximately 28% smaller than the Content Length of the XML request. The HTTP body contains all the information required to change the service instance state, the server initially determines the specific instance with the help of the instance EPR and the observer EPR, which the client sent in the request message. After determining the MobWS instance, the server invokes the ChangeState() method of the specified instance. The ChangeState() method changes the state of the instance by knowing the parameter “state”, which is the new state designated to the instance, and the parameter “previous state”, which is the previous state of the instance, in the request message from the client as shown in figure 3.8. In the request message the client wants to resume the service instance from the pause state.

![Request:](image1)

**Figure 3.8: JSON Vs XML Service Instance state change request**

After changing the state of the service instance, the ChangeState() method creates a change state response, and sends it back to the client through the Response Handler. Figure 3.9 demonstrates the comparison between JSON and XML response message.

![Response:](image2)

**Figure 3.9: JSON Vs XML Service Instance state change response**
3.3 Activity phases of the Mobile Server

To demonstrate different phases, which a server performs for manipulating a request from the client to take benefits of the MobWS, following points are discussed one by one. This discussion includes the modification in the existing system in order to facilitate JSON binding over the REST framework.

3.3.1. Request Receiving Phase.

3.3.2. Request Handling Phase.

3.3.3. Response Handling Phase.

3.3.1. Request Receiving Phase

The Request Receiving Phase shows the point where the server initially gets the request message from the client. At the Request Receiving Phase the core component used is HTTPServerThread, shown in Figure 3.10 illustrating the flow of the request corresponds to the establishment of the division phase between SOAP and REST request. In this step the type of the RPC is to be acquired by the server from the URI in the incoming request message. In order to acquire that, the server checks the message construct shown in Listing 3.1. If in the request message, the server finds the “soaprpc”, it will process the request using the SOAP architecture. On the contrary, the server executes the procedure using the REST architecture.

```java
if (msg.startsWith("POST /soaprpc")) {
    RequestHandler threadHandler = new RequestHandler(sConnection, msg);
    threadHandler.start();
    threadHandler.join();
} else {
    RESTfulHandler restfulThreadHandler = new RESTfulHandler(sConnection, msg);
    restfulThreadHandler.start();
    restfulThreadHandler.join();
}
```

Listing 3.1: Architecture Identification by HTTP Listener
Listing 3.2: Synchronous Service Request – REST Architecture

Listing 3.2 shows the URI sample sent in order to process the request using the REST architecture. It shows, the service name “CalculatorSync”, which is granted with the root element of the request message, that is the same as invoking a synchronous service in case of existing SOAP and REST architecture. After selecting the required architecture, from the REST and SOAP framework through the URI identification, without classifying rest of the parameters, such as Content-Type, the server from here onwards transfers the control to the Request Handling Phase. Letter “A” shown in Figure 3.10 illustrate the association between the Request Receiving Phase and the Request Handling Phase.

3.3.2. Request Handling Phase

The Request Handling is one of the time consuming phase as it comprises the major decisions blocks, which plays a vital role in Request Handling. RequestHandler component is the main point of concern during the Request Handling Phase and is one of the essential component in MobWS middleware as in different stages it has to acquire the necessary information in order to establish a specific decision at a particular instant. In order to conveniently understand, it is further divided into the five sub blocks as shown in Figure 3.11. Further we will discuss the request procedure and the flow between these sub blocks in detail.

1. RequestType Identifier
2. URI Identifier
3. Payload Identifier
4. ServiceType Identifier
5. RequestObject Processor

![Figure 3.11: Request Handling](image)
1. RequestType Identifier

After allocating the necessary parameter values in the RequestHandler component, the control from the Request Receiving Phase first enters into the RequestType identifier block of the Request Handling Phase, which is shown in Figure 3.12. In the RequestType identifier block, the server has to acquire the information about the type of the protocol in order to process the request using HTTP or UDP protocol. After acquiring the protocol information it will process the request message using the respective protocol. Listing 3.3 shows the code distinguishing the type of the protocol. The future discussion deals with the HTTP request messages, so further we will refer request message as HTTP. In order to accomplish some of the important tasks like parsing the URI, setting the HTTP header field and the request method, the control is passed to the RequestObject Processor.

```java
if (REQUEST_TYPE == UDP_REQUEST) {
    Process the request using UDP
} else if (REQUEST_TYPE == HTTP_REQUEST) {
    Process the request using HTTP
}
```

**Listing 3.3: Request type identification**

![RequestType Identifier](image)

**Figure 3.12: RequestType identification**
2. URI Identifier

After determining the service type from the RequestObject Processor, the server checks the URI, which is being processed by the RequestObject Processor. The server sends a fault message back to the client if it finds the "Malformed URL" as shown in Listing 3.4. If the server finds no fault in the URL, it will check for the request method. For the HTTP request method like POST and GET it will call the RequestObject processor again in order to set the payload, which is sent by the client in the request message. Figure 3.13 indicates the flow of the request message.

```java
if (!restfulRequest.tokenizeUri().equalsIgnoreCase("Malformed URL")) {
    Send fault message "Errors in Client URL"
} else {
    if (restfulRequest.getRequestMethod().equalsIgnoreCase("POST")) {
        call RequestObject Processor to set payload
    } else if (restfulRequest.getRequestMethod().equalsIgnoreCase("GET")) {
        call RequestObject Processor to set payload
    } else if (restfulRequest.getRequestMethod().equalsIgnoreCase("PUT")) {
        call RequestObject Processor to set payload
    } else if (restfulRequest.getRequestMethod().equalsIgnoreCase("DELETE")) {
        call RequestObject Processor to set payload
    }
}
```

Listing 3.4: URI Identification

![Figure 3.13: URI Identification](image-url)
3. Payload Identifier

After setting the payload in the RequestObject Processor, the control comes to the payload identifier shown in Figure 3.14. Here the server checks for the payload, if the payload does not exist, it will send a response to the client that no payload exists in the request message. On the other hand, if the payload exists, the server calls the RequestObject Processor in order to parse the payload. Listing 3.5 shows the code used by the Payload identifier.

```java
if (restfulRequest.getPayload().equalsIgnoreCase("NoPayload")) {
    No Payload in Request: sending response back to the client
} else {
    restfulRequest.parsePayload(); //if payload is set then parse
}
```

Listing 3.5: Payload Identification

![Figure 3.14: Payload Identification](image)

4. ServiceType Identifier

In ServiceType identification shown in Figure 3.15, the server acquires the information about the service type from the RequestObject Processor. The service type can be “synchronous” in case of synchronous MobWS or it can be “asynchronous” in case of asynchronous MobWS. If the server finds the service type as “synchronous” it will execute the request message using the synchronous procedure. Otherwise the request message will be processed asynchronously.

The parameter “service type” is set by the RequestObject Processor after the URI tokenizing process. The RequestObject Processor assigns it the value to synchronous or asynchronous depending on the tokens found in the URI. Such that the other component such as, ServiceType identifier can use when it is required. Listing 3.6 shows the code distinguishing both types of web services.

---

24
5. RequestObject Processor

In the RequestObject Processor, the server parses the incoming HTTP request message in order to identify the request method. For determining the request method the server uses the code shown in Listing 3.7. The HTTP request method can be one of the types, such as, POST, GET, PUT and DELETE. The server shall process the request accordingly depending on the acquired request method.
After parsing the request message and the URI in the RequestObject Processor, the server has to set the content type parameter of the HTTP header field for the response message. The server determines the content type with the help of the code shown in Listing 3.8. After determining the content type from the incoming message from the client, in case of XML request the server sets the content type as “text/xml”, while in case of JSON request the content type is “application/json”. After setting the HTTP header field in the RequestObject Processor, the control moves to the URI identifier block of the Request Handling Phase.

```
if (this.message.startsWith("POST")) {
    reqMethod = "POST";
} else if (this.message.startsWith("GET")) {
    reqMethod = "GET";
} else if (this.message.startsWith("PUT")) {
    reqMethod = "PUT";
} else if (this.message.startsWith("DELETE")) {
    reqMethod = "DELETE";
} else {
    reqMethod = "Unknown";
}
this.RequestMethod = reqMethod;
```

**Listing 3.7: Request method identification**

In the framework we are dealing with two types of MobWS - i.e. synchronous and asynchronous MobWS. Figure 3.16 shows the URI sample for both types of MobWS. To access the synchronous MobWS in the middleware the URI from the mobile client consists of one or two tokens, however to utilize the asynchronous MobWS the URI consists of more than two tokens in the request message from the mobile client.

```
int xmlstr = this.message.indexOf("Content-Type: text/xml");
if (xmlstr != -1) {
    this.contentType = "text/xml";
}
if (xmlstr == -1) {
    xmlstr = this.message.indexOf("Content-Type: application/json");
    if (xmlstr != -1) {
        this.contentType = "application/json";
    }
```

**Listing 3.8: Content type identification**
While tokenizing the URI in the URI identification, the server sets the type of the service based on the number of tokens included in the URI. For instance for the token count of one and two, the server determines the service type as synchronous and publishes it as a synchronous web service. For the token count of three, four and five, the server determines the service type as asynchronous and publishes it as an asynchronous web service. Listing 3.9 shows the code used to tokenize the URI which is sent by the mobile client.

```
UriTokenizer uriTok = new UriTokenizer(this.uri, "/");
if (uriTok.countTokens() == 5) {
    this.serviceType = "asynchronous";
} else if (uriTok.countTokens() == 4) {
    this.serviceType = "asynchronous";
} else if (uriTok.countTokens() == 3) {
    this.serviceType = "asynchronous";
} else if (uriTok.countTokens() == 2) {
    this.serviceType = "synchronous";
} else if (uriTok.countTokens() == 1) {
    this.serviceType = "synchronous";
} else {
    res = "Malformed URL";
}
```
When the RequestObject processor is called for the second time, its main task includes identification of the request Content Type and to set the payload. The server identifies each of the request messages using the code shown in Listing 3.10. It considers both the synchronous as well as asynchronous request messages. The server traverses all of the requests as shown in Listing 3.10 and after the request Content Type identification the only job for the server is to set the payload. If it does not find any message content in the request message, it sets the payload as “No payload”. Similarly the server does the same procedure in order to parse the payload. Such as, initially it identifies the request message through traversing all the developed request messages and after the identification, it will parse the payload.
### Listing 3.10: Request Content Type identification and Payload Processing

```java
int xmlstrt = this.message.indexOf("<RES>");

if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<Rq>");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<completedRq>");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<createInstanceRq>");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<listInstanceRq>");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<getPropertiesRq>");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<setPropertiesRq>");
}

if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<subscribeRq>");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<unsubscribeRq>");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<changeStateRq>");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("<stateChangedRq>");
}

if (xmlstrt != -1) {
    this.payload = this.message.substring(xmlstrt, this.message.length());
} else {
    this.payload = "NoPayload";
}

if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"Rq\"");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"completedRq\"");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"createInstanceRq\"");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"listInstanceRq\"");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"getPropertiesRq\"");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"setPropertiesRq\"");
}

if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"subscribeRq\"");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"unsubscribeRq\"");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"changeStateRq\"");
}
if (xmlstrt == -1) {
    xmlstrt = this.message.indexOf("\"stateChangedRq\"");
}
```

Figure 3.18 shows all of the asynchronous and the synchronous request message in the middleware. On the upper side of the Request Content Type Identifier all of the messages belong to JSON binding over the REST framework, while all of the lower side messages belong to XML binding over the REST framework. When the control comes from the Payload identifier to the Request Content Type Identifier, the server passes through all of the JSON as well as XML messages. Based on the content within the request which the server receives from the client, the server selects the appropriate request message and memorizes it for the invocation of the respective synchronous/ asynchronous MobWS. After the identification the control moves to the Service Type Identifier.
### 3.3.3. Response Handling Phase

Response Handling Phase shows the point where the mobile server gets the response from the MobWS. At the Response Handling Phase the core component used is RESTfulResponse Handler. The flow of the response corresponds to the establishment of the division phase between JSON and XML response, with the help of the code shown in Listing 3.11. In this step the Response Content Type is to be acquired from the incoming response message. If in the response message, the server finds the Content Type as JSON, it will send the JSON response message back to the mobile client after attaching the HTTP header with the response body. On the contrary, the server executes the procedure, as attaching the HTTP header with the response body, which is designed for XML response message to the mobile client.
Listing 3.11: JSON Vs XML Response Handling Implementation
Chapter 4

Performance Evaluation of Mobile Web Services

4.1. Overview
In this chapter we will discuss the performance of the JSON enabled MobWS middleware. The JSON framework is introduced over the existing REST architecture, where the architecture is capable to host the synchronous and asynchronous MobWS. Hence, the evaluation is done to compare the performance in both the JSON and XML enabled REST architectures for both types of Web Services.

4.2. Performance Evaluation Criteria
Initially in this section the environment is demonstrated which is used to perform the evaluation. Secondly we will discuss different ways of testing foundation. After providing the description about the foundation, it would be convenient to understand various test cases, which are presented further in the chapter.

4.2.1. Testbed
While evaluating the performance of the REST architecture, table 4.1 presents the system specification, which was used for the Performance Evaluation. Table 4.1 shows most of the prominent parameters, which plays a vital role for better performance results.

<table>
<thead>
<tr>
<th>System Parameters</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>Hewlett-Packard</td>
</tr>
<tr>
<td>Model</td>
<td>HP Pavilion dv6700 Notebook PC</td>
</tr>
<tr>
<td>Processor</td>
<td>AMD Turion(tm) 64*2 Mobile Technology TL-60</td>
</tr>
<tr>
<td>Speed</td>
<td>2.00 GHz</td>
</tr>
<tr>
<td>Memory (RAM)</td>
<td>3.00 GB</td>
</tr>
<tr>
<td>Operating System</td>
<td>Windows Vista™ Home Premium – Service Pack 2, 32-Bit</td>
</tr>
</tbody>
</table>

Table 4.1: System Specification

4.2.2. Performance Metric
In order to demonstrate the Performance Evaluation comparison between the existing system and the modified system, a well organised overview of the evaluated parameters is shown in Table 4.2. It includes all the measured latencies (time), while evaluating the performance of the REST architecture, using XML and JSON messaging constructs, for the synchronous as well as the asynchronous MobWS.
4.2.2. Performance Evaluation Parameters

To have a fastidious understanding of the parameters used for the Performance Evaluation, reading the following details would be useful.

4.2.2.1. Payload Formation

It basically describes the number of bytes (octets) used to construct the request messages by the mobile client. It is clearly evident from figure 4.3 and 4.4, that for the same request the number of bytes differs, when the mobile client constructs XML and JSON request messages.

4.2.2.2. Server Processing Time

The total time consumed by the mobile server to process the request from the mobile client. Means the time from the point, where the server initially receives the incoming request to the point just after, where the server sends the response back to the mobile client.

4.2.2.3. Payload Parse Time (Server)

It is the time consumed by the parser module, in the mobile server, for parsing the payload (HTTP body) in the HTTP request message. It does not include the parsing time, used to parse the HTTP header fields in the request message.

4.2.2.4. Service Invocation Time (Server)

The time required by the mobile server to invoke the Mobile Web Service requested by the mobile client. We consider the difference between two reference points in order to calculate Service

<table>
<thead>
<tr>
<th>Performance Evaluation Parameters</th>
<th>REST Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SYN MobWS</td>
</tr>
<tr>
<td>Payload Formation</td>
<td></td>
</tr>
<tr>
<td>Payload Parse Time (Server)</td>
<td></td>
</tr>
<tr>
<td>Service Invocation Time (Server)</td>
<td></td>
</tr>
<tr>
<td>Response Processing Time (Server)</td>
<td></td>
</tr>
<tr>
<td>Request Preparation Time (Client)</td>
<td></td>
</tr>
<tr>
<td>Request Transport Module Time (Client)</td>
<td></td>
</tr>
<tr>
<td>Time Difference</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Performance Evaluation Parameters
Invocation Time. The time at the point where the server initially receives the request message is our first point of reference and our second point of reference is the point just before invoking the MobWS by the mobile server.

4.2.2.5. Response Processing Time (Server)

The time used for processing the response message by the mobile server, before sending the response back to the mobile client. While measuring it, the difference between the two reference points, is also used here. First point is the time where the mobile server receives the response from the requested MobWS and the second point of reference is the time just before sending the response message back to the mobile client.

4.2.2.6. Request Preparation Time (Client)

The time consumed by the mobile client for preparing the request message. Its measurement is not much effective, when the request size is small. However as the message size increases dramatically, request preparation time grows exponentially and gives a valuable comparison overview between XML and JSON message construction.

4.2.2.7. Request Transport Module Time (Client)

The time spent by the mobile client before sending the request to the mobile server. This time is measured by using the difference between two time stamps. First one is the time while initiating the request preparation and second one is the time just before sending the request to the mobile server.

4.2.2.8. Time Difference

Time difference is the time which concludes most of the dominant Performance Evaluation results by comparing them in a way that put forward the most effective parameter (processing time).

All of the above discussed performance evaluation parameters are measured simultaneously to show and compare the results with different payload size of the request messages when a XML/JSON message format is transmitted by the mobile client to the mobile server.

4.3. Scenarios Analysed

The existing REST architecture is capable to transmit XML messages between the mobile client and server, in order to host the synchronous as well as asynchronous MobWS. The existing middleware is modified by redefining the messaging format to the JSON messaging format. Now to host the MobWS, the REST middleware becomes capable to use both of the XML as well as JSON messaging format. We have used the following test scenarios in order to compare the modified middleware with the earlier version of the middleware.

We have defined two different situations (test scenarios) for the Performance Evaluation of the REST middleware. First one show the test, when the XML messages are used between client and server. Second scenario deals with the situation when the REST architecture uses JSON as a messaging format.
4.3.1. Test Scenario 1: XML Binding

This test scenario examines the situation when XML is used in the REST middleware as a messaging format. We have further divided this scenario into two sub blocks to put forward an evaluated comparison overview between the synchronous and the asynchronous request messages.

4.3.1.1. Synchronous Incremental Test

This test shows the performance of the mobile server while invoking the synchronous MobWS (CalculatorSync) with incremental XML request messages from the client. Here word “Incremental” means increment in the number of elements by a factor of $2^n$, while payload formation for the request message. For example for the second request, the request massage includes four elements; similarly for the third request, the request message includes eight elements and so on. Figure 4.1 shows the XML message which was sent by the mobile client for the second request. Figure 4.3 shows the Payload Formation for all the twelve request messages, which we used for the Performance Evaluation of the mobile server.

4.3.1.2. Asynchronous Incremental Test

In this test we analyzed the performance of the mobile server while dealing with asynchronous MobWS (Calculator) with similar incremental pattern of the XML request messages like we used for Synchronous Incremental Test. Figure 4.5 shows the Payload Formation for all the twelve request messages, which we used while evaluating the Performance of the mobile server.

4.3.2. Test Scenario 2: JSON Binding

This test shows the analyses when JSON messages are used in the REST architecture during the Client-Server communication. To compare the Performance Evaluation results for the synchronous and asynchronous MobWS we have separated this test into the following sub blocks.

4.3.2.1. Synchronous Incremental Test

This test shows the performance of the server while invoking the synchronous MobWS (CalculatorSync) with incremental JSON request messages from the client. Here word “Incremental” means increment in the number of elements by a factor of $2^n$, while payload formation for the request message. For example for the second request, the request massage includes four elements; similarly for the third request, the request message includes eight elements and so on. Figure 4.2 shows the JSON message which was sent by the mobile client for the second request. Figure 4.4 shows the Payload Formation for all the twelve request messages, which we used for the Performance Evaluation of the server.

4.3.2.2. Asynchronous Incremental Test

In this test we analyzed the performance of the server while dealing with asynchronous MobWS (Calculator) with similar incremental pattern of the JSON request messages like we used for Synchronous Incremental Test. Figure 4.6 shows the Payload Formation for all the twelve request messages, which we used while evaluating the Performance of the mobile server.
Synchronous XML Second Request

POST /CalculatorSync HTTP/1.1
Content-Type: text/xml
Content-Length: 126
Host: 127.0.0.1:9091

<Request>
  <ContextData>
    <CalcData>
      <value1/>
      <value2/>
      <value3/>
      <value4/>
    </CalcData>
  </ContextData>
</Request>

Figure 4.1: Synchronous XML message construct

Synchronous JSON Second Request

POST /CalculatorSync/add HTTP/1.1
Content-Type: application/json
Content-Length: 77
Host: 127.0.0.1:9091

{"Request":
  {"ContextData":
    {"CalcData":
      {"value1":,
       "value2":,
       "value3":,
       "value4":}}
}}

Figure 4.2: Synchronous JSON message construct

Synchronous XML Request Formation

![Synchronous XML Request Formation Graph](image)

Figure 4.3: Synchronous XML request formation

Synchronous JSON Request Formation

![Synchronous JSON Request Formation Graph](image)

Figure 4.4: Synchronous JSON request formation

Asynchronous XML Request Formation

![Asynchronous XML Request Formation Graph](image)

Figure 4.5: Asynchronous XML request formation

Asynchronous JSON Request Formation

![Asynchronous JSON Request Formation Graph](image)

Figure 4.6: Asynchronous JSON request formation
4.3.3. Evaluation Summary

Table 4.3 shows the summary of the Test Scenario 1 and the Test Scenario 2. We have compared the XML binding with the JSON binding over the REST architecture for the synchronous and the asynchronous MobWS. In order to analyse the performance of the mobile server we have examined the request messages from the client with varying message size as shown in Table 4.3. Short description of the parameters used in Table 4.3 is as follows.

- **Test Case:** Name of the test case.
- **Service name:** Middleware function/module used for the specific test case.
- **Request Number (n):** Shows the number of measurements.
- **Elements/Request (2^n):** Number of Elements sent for a particular request.
- **Bytes/Request:** Bytes used for representing the respective Number of Elements.

<table>
<thead>
<tr>
<th>Test Cases</th>
<th>Service name</th>
<th>Request Number (n)</th>
<th>Elements/Request (2^n)</th>
<th>Bytes / Request XML Binding</th>
<th>Bytes / Request JSON Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous Incremental Test</td>
<td>CalculatorSync</td>
<td>1</td>
<td>2</td>
<td>84</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>4</td>
<td>126</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>8</td>
<td>210</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>16</td>
<td>378</td>
<td>245</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>32</td>
<td>714</td>
<td>469</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>64</td>
<td>1386</td>
<td>917</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>128</td>
<td>2730</td>
<td>1813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>256</td>
<td>5418</td>
<td>3605</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>512</td>
<td>10794</td>
<td>7189</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>1024</td>
<td>21546</td>
<td>14357</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>2048</td>
<td>43050</td>
<td>28693</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>4096</td>
<td>86058</td>
<td>57365</td>
</tr>
<tr>
<td>Asynchronous Incremental Test</td>
<td>Calculator</td>
<td>1</td>
<td>2</td>
<td>354</td>
<td>216</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>4</td>
<td>407</td>
<td>244</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>8</td>
<td>511</td>
<td>299</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>16</td>
<td>719</td>
<td>412</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>32</td>
<td>1135</td>
<td>636</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>64</td>
<td>1967</td>
<td>1084</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>128</td>
<td>3630</td>
<td>1980</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>256</td>
<td>6950</td>
<td>3772</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>512</td>
<td>13611</td>
<td>7356</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>1024</td>
<td>22818</td>
<td>14524</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>2048</td>
<td>45346</td>
<td>28860</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>4096</td>
<td>90401</td>
<td>57532</td>
</tr>
</tbody>
</table>

**Table 4.3:** Test Summary
4.4. Message Size Comparison

Figure 4.7 demonstrates the message size comparison (in bytes) for the XML and JSON request messages which was sent by the mobile client to the mobile server. We have enhanced the REST middleware in the way that the synchronous request can be of XML or JSON format. Both of the formats correspond to the same information; however the representation of the request message is different because of the difference in the basic messaging structure. Such as XML uses tags and content duplication to describe the data which increases the message size whereas JSON contains no tags and no content duplication which results in the transmission of less data between the client and the server. In the figure 4.7 for all the 12 requests, the blue bar represents the number of bytes consumed using the XML format, where as the red bar shows the JSON message size.

For all the request messages it is important to mention here that the XML message size always remains greater than the JSON message size. Also we have compared XML and JSON message size and we have observed that by using JSON as a messaging format we have around 33 % reduction in the message size as shown in Figure 4.7 for all the requests with the yellow bar which results in the memory consumption. Hence our initial assumption seems to be correct that by using XML, a lot of resources might be over utilized, such as the mobile node’s memory and processor. So by using JSON as the messaging format now we are able to save the over utilized resources.

![Figure 4.7: Synchronous XML Vs JSON Message Size Comparison](image)

Similarly for the REST architecture we have also enhanced the middleware the same way that the asynchronous request messages can be of XML or JSON format. Figure 4.8 shows the asynchronous requests with incremental message size comparison, for the XML and JSON requests, which was transmitted by the mobile client to the mobile server to invoke the asynchronous MobWS. For the asynchronous requests we have achieved the same goals like we achieved for the synchronous request messages, that we have reduced the over utilization of the resources (mobile node’s memory and processor) which are utilized larger, by using the XML request messages. We have compared the XML and JSON message size for all the asynchronous requests and we have found that there is approximately 36 % drop in the message size when JSON is used. The comparison is shown with the yellow bar in Figure 4.8.
4.5. Processing Time Analyses

We have separately analysed the processing time for the invocation of both types of MobWS, named as Synchronous MobWS and Asynchronous MobWS. The main factor which forced us to analyse them separately is, both types uses different components and procedures to invoke the respective MobWS.

With the enhancement of the REST architecture now it is possible that the request messages can be of XML or JSON format. In short we have introduced XML as well as JSON Binding over the REST architecture. In the previous section we have compared both of the XML and JSON request messages in the context of message size, which was sent by the Mobile Client to the Mobile Server. In the following section we are going to compare both of the XML and JSON request messages in terms of processing time.

Processing Time

In this section we have analyzed and compared the Processing Time for the Synchronous and Asynchronous MobWS using the JSON and XML Binding over the REST architecture. In order to compare the performance, we have separately sent the XML messages and the JSON messages. And we obtained the results separately. However to compare them more efficiently we have put forwarded the Processing Time analyses for both of the messaging formats (i.e. XML and JSON) simultaneously.

1. Server Processing Time

To evaluate the performance of the Mobile Server, we have calculated the overall Processing Time of the mobile server, known as Server Processing Time shown in Figure 4.9. For the Performance Evaluation in every request we have exponentially increased the message size by a factor of $2^n$, which was sent to the Mobile Server. And we have calculated the respective Processing Time for all of the requests shown in Figure 4.9.

From the obtained results we have observed that as we increase the message size, the Processing Time also increases exponentially. However in case of XML Binding we have seen a dramatically increase in the Processing Time as compared to the Processing Time evaluated using JSON Binding.
over the REST architecture. In Figure 4.9 at the 12th request, the same data is transmitted to the mobile server only the messaging format is different but we have observed a significant difference in the Processing Time. The XML Processing Time is 19672 milliseconds that is 95% higher than the JSON Processing Time which is 984 milliseconds.

![Server Processing Time](image)

**Figure 4.9**: Synchronous XML Vs JSON Server Processing Time Comparison

The Mobile Server servers ~95% much better using JSON messaging format because the Payload Parse Time required to parse the JSON message at the server is small, as the parser only have to tokenize the JSON string in order to parse the JSON data. Whereas in case of XML the parser have to parse the XML message element by element over the entire XML tree, which is computationally expensive and results in an increase in the XML Payload Parse Time. Another important factor which mainly influences the Server Processing Time is the Response Processing Time. The Response Processing Time, using JSON messaging format, is small, as the Mobile Server only has to store the JSON message in a string variable which will be transmitted to the Mobile Client. Whereas in case of XML, before transmission, the Mobile Server has to write the XML data element by element over the Byte Array Output Stream using the XML writer, which consumes a lot of time for the response preparation. Figure 4.10 shows the processing time of the mobile server using XML and JSON binding over the REST architecture for asynchronous MobWS.

![Server Processing Time](image)

**Figure 4.10**: Asynchronous XML Vs JSON Server Processing Time Comparison
2. Payload Parse Time

For the synchronous incremental test, the increase in the payload size is demonstrated in Table 4.3. With the increase in the request number the payload size is also increased by a factor of 2⁴. Such as for the first request only two elements are transmitted, for the second request four elements are transmitted and so on. The time required to parse the entire payload message by the mobile server is known as payload parse time.

![Payload Parse Time](image1.png)

**Figure 4.11:** Synchronous XML Vs JSON Payload Parse Time Comparison

![Payload Parse Time](image2.png)

**Figure 4.12:** Asynchronous XML Vs JSON Payload Parse Time Comparison

From the evaluated results shown in figure 4.11 and 4.12, we have observed that the XML payload parse time remains always greater than the JSON payload parse time. As demonstrated for the 12th request in figure 4.11 the JSON payload parse time is 62% smaller than the XML payload parse time because in case of JSON the parser only has to tokenize the JSON string in order to parse the JSON data, whereas in case of XML the parser have to parse the XML message element by element for the entire XML tree. With the exponential increase in the number of elements the tree size also increases in the same manner which makes the XML computation more complicated and results in an increase in the XML payload parse time.
3. Service Invocation Time

Figure 4.13 and figure 4.14 shows the service invocation time of the mobile server for the synchronous and asynchronous MobWS respectively. Using the REST architecture the mobile server invokes the synchronous MobWS based on the synchronous request from the mobile client. In context of service invocation, to evaluate the performance of the mobile server, we have compared the synchronous XML versus JSON service invocation time. The request size in all the requests shown in figure 4.13, increases in the same manner as we have discussed in Section 4.3.1.1 and 4.3.2.1.

![Figure 4.13: Synchronous XML Vs JSON Service Invocation Time Comparison](image1)

We have observed from the evaluated results shown in figure 4.13 that initially for the 1st request the Service Invocation Time is higher for both the XML and JSON relatively to the Service Invocation Time calculated until the 9th request. The possible reason is initially the Service Invocation Time also includes the scheduling time of the operating system.

![Figure 4.14: Asynchronous XML Vs JSON Service Invocation Time Comparison](image2)

It can also be evaluated from the figure 4.13 that for all the requests the Service Invocation Time remains higher for the XML as compared to the JSON. For example in the 12th request using the JSON
messaging format the server performs the service invocation approximately 56% earlier as compared to the XML messaging format. The reason is before the service invocation, the mobile server has to acknowledge the type of the service to invoke. To get the Service Type information the mobile server has to parse whole of the request message. Because XML has a tree structured element formation which makes it’s processing more difficult and therefore consumes a lot of time. While the JSON has a string format and the parser only has to tokenize the JSON string in order to parse it, which consumes less processing time as compared to the XML processing time.

4. Response Processing Time

Response processing time, explained in section 4.2.2.5, plays a vital role in increasing the overall server processing time using XML messaging format as compared to the JSON messaging format. We have compared the response processing time for both the XML and JSON and the evaluated outcome is demonstrated in figure 4.15. During the evaluation we have followed the same procedure that with the increase in the request number the request message size also increases exponentially by a factor of $2^n$.

43

![Figure 4.15: Synchronous XML Vs JSON Response Processing Time Comparison](image)

We have observed from the results shown in figure 4.15 that for all the requests the response processing time always remain greater for the XML as compared to the JSON. For the final requests we have observed that the response processing time increases exponentially using XML messaging format. For example for the 12th request shown in figure 4.15 XML has approximately 99% greater response processing time as compared to the JSON. Because in case of XML, before transmission, the mobile server has to write the XML data, element by element, over the Byte Array Output Stream using the XML writer, which consumes a lot of time for the response preparation. And as the request size increases exponentially, its response processing time also increases exponentially.

5. ServiceRqTransport Time

In asynchronous request-response processing, a mobile client do not have any idea of after how much time the response message shall arrive from the MobWS which is invoked by the mobile server. In order to evaluate and compare the performance of multiple content types (JSON Vs XML) we have calculated the ServiceRqTransport processing time shown in Figure 4.16.

43
From the evaluated results in Figure 4.16 we have observed that ServiceRqTransport time for XML remains always greater than ServiceRqTransport time for JSON. Because after receiving the request (CompletedRq()) message from the RESTful Service Notification class the RESTful Observer class has to send back the response (CompletedRs()) message. As XML response preparation takes longer time, so ServiceRqTransport time in case of XML is approximately 98% greater as compared to the time consumed by JSON as shown in Figure 4.16.

6. Time Difference

Figure 4.17 demonstrates the overall processing time of the mobile server, while evaluating the XML binding over the REST architecture. The server processing time includes the service invocation time and the response processing time. We considered the payload parse time while evaluating the service invocation time. The description of the mentioned processing time has been explained in sections 4.2.2.

Figure 4.16: Asynchronous XML Vs JSON ServiceRqTransport Time Comparison

Figure 4.17: Synchronous XML Time Difference
The evaluated results are the outcome of the incremental request messages by a factor of $2^n$, which are sent by the mobile client to the mobile server. As shown in Table 4.3 in the 1st request 84 bytes are sent using XML binding over the REST architecture. With the increase in the request number the message size also increases. For example as shown in Figure 4.17, in the 12th request, 86058 bytes are sent to the mobile server.

From the evaluated results shown in Figure 4.17 we have observed that the XML messaging format consumes a lot of time, such that the mobile server takes approximately 20,000 milliseconds to process the 12th request. We have also observed that the most prominent factor which greatly effects the overall server processing time is the response processing time. Because in case of XML, before transmission, the mobile server has to write the XML data, element by element, over the Byte Array Output Stream using the XML writer, which consumes a lot of time for the response preparation. And as the request size increases exponentially, its response processing time also increases exponentially. For example in the 11th request shown in Table 4.3 when the XML message size is 43050 bytes, its processing time is approximately 4,500 milliseconds shown in Figure 4.17, as the message size increases exponentially to 86058 bytes in the 12th request, its processing time dramatically increases to approximately 20,000 milliseconds. The other factor, with minute effects, includes the payload parse time and the service invocation time. Another important thing to notice is for the same request message (12th request) with only the change in the Content-Type, the XML format consumes 86058 bytes as shown in Table 4.3 and the JSON format consumes 57365 bytes as shown in Table 4.3, which results in saving approximately 34% of memory using the JSON format as compared to the XML format.

![Figure 4.18: Synchronous JSON Time Difference](image)

Figure 4.18 demonstrates the overall processing time of the mobile server, while evaluating the JSON binding over the REST architecture. In order to have an appropriate comparison between JSON and XML binding over the REST architecture, we have evaluated the processing time in case of JSON shown in Figure 4.18 as compared to the XML shown in Figure 4.17. Also the increment in the message size follows the same manner as discussed in Table 4.3.

From the evaluated results shown in Figure 4.18 we have observed that a lot of time is saved by using JSON as a messaging format, such as the mobile server takes less than 1,000 milliseconds to process the 12th request. On the other hand using the XML as the messaging format shown in Figure
4.17, the mobile server takes approximately 20,000 milliseconds to process the 12th request. This demonstrates, using XML as a messaging format, consumes 20 times greater processing time as compared to the JSON. Similar results are achieved for the asynchronous MobWS as compared in Figure 4.19 and 4.20 for XML and JSON respectively. And an important point to mention here is, for all the requests, we have the same request message only the Content-Type (JSON/XML) is different.

The most prominent factor which reduces the overall Server Processing Time using the JSON messaging format is the response processing time. The response processing time, using JSON messaging format, is small, because the mobile server only has to store the JSON message in a string variable which will be transmitted to the mobile client. Whereas in case of XML, before transmission, the mobile server has to write the XML data element by element over the Byte Array Output Stream using the XML writer, which consumes a lot of time for the response preparation.

![Figure 4.19: Asynchronous XML Time Difference](image1)

![Figure 4.20: Asynchronous JSON Time Difference](image2)
4.6. Software Size

The JSON Enabled REST framework is the enhancement of the existing REST framework. The software is developed for both the synchronous (short-lived) and asynchronous (long-lived) architecture of the REST framework. Table 5.4 demonstrates the size of the REST packages, the size of the JSON packages and overall package size of the JSON Enabled REST framework. The JSON Package size is 131 KB, which is approximately 16 % increment in the size of existing framework. The total size of the JSON Enabled REST framework is 824.5 KB. The package sizes which are shown in Table 5.4 show the non-obfuscated size of the software. However using the obfuscation techniques we can reduce the software size to much lower value. This is achieved by using shorter names for classes, methods and fields in JAVA which result in decrease in the software size.

<table>
<thead>
<tr>
<th>REST Packages</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnRestful</td>
<td>442.0</td>
</tr>
<tr>
<td>cnRestful.util</td>
<td>38.0</td>
</tr>
<tr>
<td>cnRestful.Server</td>
<td>43.2</td>
</tr>
<tr>
<td>cnRestful.Services</td>
<td>38.3</td>
</tr>
<tr>
<td>cnRestful.ASAPTypes</td>
<td>132.0</td>
</tr>
<tr>
<td>REST Total Size (Existing Framework)</td>
<td>693.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JSON Packages</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.json.me</td>
<td>97.6</td>
</tr>
<tr>
<td>org.json.me.util</td>
<td>22.7</td>
</tr>
<tr>
<td>org.json.me.test</td>
<td>10.7</td>
</tr>
<tr>
<td>JSON Total Size</td>
<td>131.0</td>
</tr>
</tbody>
</table>

| JSON Enabled REST Size (Enhanced Framework) | 824.5 |

Table 4.4: Package Size of JSON Enabled REST Framework
Chapter 5

Theses

- A generic lightweight JSON messaging format has been developed and integrated with the existing REST framework in order to access the Mobile Web Services (MobWS).

- JSON messaging format has been implemented for both the synchronous (short-lived) and asynchronous (long-lived) architecture of the MobWS middleware.

- The development of multiple content type feature over the MobWS middleware broaden the scope of hosting Web 2.0 services on the mobile devices to address diverse types of mobile clients.

- The development of JSON messaging format is also used to bring IMS client side compatibility to the MobWS middleware using Ericsson Mobile Java Communication framework (MJCF).

- To identify and compare the performance of multiple content types (JSON Vs XML) the REST framework is extensively evaluated for invoking both synchronous and asynchronous MobWS.

- The JSON Package size is 131 KB, which is approximately 16 % increment in the size of existing middleware. The total size of the JSON Enabled REST framework is 824.5 KB.
Chapter 6
Conclusions and Outlook

7.1 Conclusions

Because of the increasingly distinct capabilities of Web Services clients, today Web 2.0 supports Web Services with multiple content types to address diverse types of clients. The implementation of the JSON framework is an efficient way of providing multiple content types feature to the existing REST based Mobile Web Services MobWS middleware for the synchronous and asynchronous mobile architecture. In order to achieve this task, the framework is analysed and enhanced by developing the JSON modules, which are defined in the ECMA-262 standard, and are optimized in conformance with the analysed architecture. Outcome of this research has produced a generic lightweight JSON framework. This development makes the enhanced MobWS middleware compatible with the new developing technologies. Such as the latest versions of the Web Browsers are now providing native, safe support for encoding and decoding of JSON data, so the browsers can manipulate large amount of JSON data much more efficiently than they can manage large amount of XML data. The reason is a growing number of Web Services return simple data structure, serialized as JSON string and JSON string is nothing but a tightly constrained JavaScript program which makes it a lot easier for a Web Browser to get a JavaScript data structure from a JSON data as compared to the XML data. In this way the Web Browsers uses the same JavaScript interface to parse the JSON data which is to be consumed by the clients. Additionally JSON binding developed in this work is used to enable service consumption in the IMS. This work is done by developing IMS client side modules in the middleware using Ericsson’s Mobile Java Communication Framework (MJCF). Finally the performance analyses have been done to evaluate and compare the impact of JSON Vs XML Binding over the synchronous as well as for the asynchronous server architecture. Thus, the architectural capability of the MobWS middleware has been customized to select the appropriate content type to handle and process the client request.

7.2 Outlook

REST based Mobile Web Server provides the services, such as the instant messaging to share the information, such as SMS. However to deliver large data files, such as an audio or a video clip, an image, among the mobile users; the existing framework has the limitation that the session based instant messaging is currently not implemented over the framework, which restricts the mobile clients to exchange large size of files in a session. Development of session based instant messaging will make it possible for the mobile peers to communicate in a near real time and it will also increase the Quality of Service. The future development will include the usage and study of SIP, SDP, MSRP and TCP protocols for the actual media transmission over the IMS network.
List of Figures

2.1: JSON data structure..........................................................12
2.2: JSON Format..................................................................12
2.3: XML Format..................................................................12
3.1: REST Mobile Web Services Middleware..........................14
3.2: XML messaging construct..................................................15
3.3: XML element allocation in memory....................................15
3.4: JSON Vs XML over REST Binding.....................................16
3.5: JSON Vs XML Synchronous Response..............................17
3.6: JSON Vs XML Create Instance Request..............................18
3.7: JSON Vs XML Create Instance Response............................18
3.8: JSON Vs XML Service Instance state change request..........19
3.9: JSON Vs XML Service Instance state change response........19
3.10: RPC identification.........................................................20
3.11: Request Handling..........................................................21
3.12: RequestType identification...............................................22
3.13: URI Identification..........................................................23
3.14: Payload Identification.....................................................24
3.15: ServiceType Identification...............................................25
3.16: URI Sample...................................................................27
3.17: RequestObject Processor...................................................28
3.18: Request Content Type Identification...............................29
4.1: Synchronous XML message construct...............................36
4.2: Synchronous JSON message construct...............................36
4.3: Synchronous XML request formation................................36
4.4: Synchronous JSON request formation...............................36
4.5: Asynchronous XML request formation...............................36
4.6: Asynchronous JSON request formation...............................36
4.7: Synchronous XML Vs JSON Message Size Comparison.......38
4.8: Asynchronous XML Vs JSON Message Size Comparison.......39
4.9: Synchronous XML Vs JSON Server Processing Time Comparison................................................38
4.10: Asynchronous XML Vs JSON Server Processing Time Comparison..................................................40
4.11: Synchronous XML Vs JSON Payload Parse Time Comparison..................................................41
4.12: Asynchronous XML Vs JSON Payload Parse Time Comparison..................................................41
4.13: Synchronous XML Vs JSON Service Invocation Time Comparison..................................................42
4.14: Asynchronous XML Vs JSON Service Invocation Time Comparison..................................................42
4.15: Synchronous XML Vs JSON Response Processing Time Comparison..................................................43
4.16: Asynchronous XML Vs JSON ServiceRqTransport Time Comparison..................................................44
4.17: Synchronous XML Time Difference....................................44
4.18: Synchronous JSON Time Difference....................................45
4.19: Asynchronous XML Time Difference....................................46
4.20: Asynchronous JSON Time Difference....................................46

50
### List of Tables

<table>
<thead>
<tr>
<th>4.1: System Specification</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2: Performance Evaluation Parameters</td>
<td>33</td>
</tr>
<tr>
<td>4.3: Test Summary</td>
<td>37</td>
</tr>
<tr>
<td>4.4: Package Size of JSON Enabled REST Framework</td>
<td>47</td>
</tr>
<tr>
<td>Notation</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transport Protocol</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>JSON</td>
<td>Java Script Object Notation</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>SOC</td>
<td>Service Oriented Computing</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>IMS</td>
<td>IP Multimedia Subsystem</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>SDP</td>
<td>Session Description Protocol</td>
</tr>
<tr>
<td>MSRP</td>
<td>Message Session Relay Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>MobWS</td>
<td>Mobile Web Services</td>
</tr>
</tbody>
</table>


[13] Introducing JSON, Published: http://www.json.org/