An Empirical Evaluation of Web System Access for Smartphone Clients

Scott Fowler*, Katrin Hameseder and Anders Peterson
Linköping University
Department of Science and Technology
SE-601 74 Norrköping, Sweden
*Corresponding author: Email: scott.fowler@liu.se

Abstract—As smartphone clients are restricted in computational power and bandwidth, it is important to minimise the overhead of transmitted messages. This paper identifies and studies methods that reduce the amount of data being transferred via wireless links between a web service client and a web service. Measurements were performed in a real environment based on a web service prototype providing public transport information for the city of Hamburg in Germany, using actual wireless links with a mobile smartphone device. REST based web services using the data exchange formats JSON, XML and Fast Infoset were evaluated against the existing SOAP based web service.

Index Terms—SmartPhones, XML, iPhones, Ubiquitous, Measurements, Web Service

I. INTRODUCTION

The Internet growth over recent decades has induced increasing demands for high speed and ubiquitous access. With the introduction of smartphones (e.g. iPhone), the market for powerful mobile devices has exploded. Cisco predicted that mobile traffic would increase by a factor of 39 times between 2009 and 2014 [1]. The recognition of computing paradigms varies with time. The emerging interests of iPhone was depicted using the Google search trends from 2006 to February 2012 (Figure 1). The y-axis represents the number of searches that have been done for iPhone relative to the total number of searches done using Google search from 2006 to February 2012. The values are normalized into a scale of 0 to 100, and the calculations are detailed in the Google web site. The interests of other smartphone brands are increasing in a similar way.

Vendors offering state-of-the-art smartphones have these devices running on different operating systems, which often leads to compatibility problems when porting desktop services. One possibility to avoid such problems is to use web services that act as mobile clients and provide the features of a desktop service for mobile devices independently of the operating system or client application of the mobile device. However, limitations in the smartphones’ CPU main memory and RAM strongly bound the capabilities for such services. Further, as mobile devices, and particularly smartphones, only have limited power supplies, it is of importance to design the service in a way that ensures that less power as possible is consumed, and that the processed data is kept in case of power loss.

A. Contribution

The focus of this paper is on web services, and methods for improving their usability in mobile networks. Mobile devices usually connect to web services over wireless connections. These wireless links lead to higher response times due to limited bandwidth and high latency compared to wired connections. This paper identifies and studies methods that reduce the amount of web service data being transferred via wireless links to and from a web service client. Today the limited bandwidth in wireless connections often is a bottleneck for the execution time of a web service from a mobile device. Hence, reducing the amount of data being transferred makes the use of the web service from a mobile client faster. As an additional benefit, the communication cost will be reduced on links that typically are charged on a per byte basis.

The contribution of this paper is an evaluation of response time and total bytes used to access the web services from mobile units with limited computational and memory resources, e.g. mobile smartphones. Measurements were performed in a real environment based on a web service prototype providing public transport information for the city of Hamburg in Germany.
An evaluation of the existing SOAP (Simple Object Access Protocol) based web service has been done in order to determine how to enable efficient access for smartphone clients. The current usage of the SOAP leads to a high amount of overhead in the request and response messages and is not applicable for smartphone clients. Comparisons of SOAP against REST using the data exchange formats JSON, XML and Fast Infost are developed to verify that the use of a REST (Representational state transfer) web service as well as the choice of the data format leads to improvements in terms of:

- transferred data volume
- serialisation/deserialisation time
- request–response time

The reminder of this paper is organized as follows: In the next section, we present work related to our contribution. Section II describes the web services. In Section III and IV, we present the data exchange formats and discuss serialisation/deserialisation aspects of them. In Section V, the implementation is described, and Section VI contains the results. Section VII concludes the paper.

B. Related Work

Most of today’s web services use XML for the representation of the data that has to be transferred between the web server and the mobile client [3]. The large amount of overhead data that is involved when processing XML is a problem for mobile devices. One possibility to make the access of information more efficient is to use techniques that reduce the overhead of XML. Jonsrud et al. [4] analysed the efficiency of usage of different XML compression techniques. The first compression technique which they evaluated, is known as generic compression and is characterised by the fact that compressed XML messages have to be decompressed in order to be processed by the application. The second compression technique that was considered is XML-aware compression, where the information structure and the hierarchy of the XML document is preserved after compression. The performed analysis shows that the best compression rate was achieved when the use of compression was dependent on the message size. However, a wireless mobile device with constantly changing compression rate resulted in an increased overhead and, hence, a more demanding processing on the mobile client.

Another approach to deal with the limitations of mobile devices and the problems with SOAP based web services was proposed by Schmutzler et al. [5]. Their performance analysis of SOAP-XML, WAP Binary XML based on SOAP, and JXTA used measurements of the data traffic, transmission delay and error rate (failed requests). The results show that the use of XML-based SOAP web services is very inefficient in terms of data traffic and transmission delay. In contrast, the use of WAP Binary XML based on SOAP resulted in smaller messages that are transmitted over the network, which in turn, resulted in a lower transmission delay. JXTA is more efficient in terms of transmission delay when messages with larger size are sent. However, fixed sized messages are not guaranteed in wireless mobile network.

The research of Lai et al. [6] focuses on SOAP web services and the analysis of the use of the different SOAP bindings HTTP, TCP and UDP. The results showed that the use of the SOAP bindings HTTP and TCP provided higher overhead with less throughput compared to UDP. However, the provided features depend largely on the used platform. This often leads to compatibility problems when porting desktop applications to mobile devices.

The reduction of redundancies in a stream of web service calls [8]–[10] have been investigated so as to address web service performance overhead for mobile clients. These papers have focused on XML with SOAP instead of other web services such as Infost and JSON. Shum et al. [11] attempted to demonstrate a correlation between the use of laptops and student motivation by studying the web browsing behaviour of students. This study did not show the type of web service being investigated, but focused on the correlation between use of laptops in an educational settings. Compression is one way of dealing with the problem of large message sizes of web Services [12]. Studying tradeoff between compression/decompression and CPU time would provide better insight in handling large network. In the future work, we would consider investigating the use of compression for studying web service performance. While [13] focused on the various web service providers and the performance of on smartphones. We described identification and analysis of factors that impact user’s perceived performance of web server network applications on smartphones.

In this paper, we evaluate the overall performance for different ways of accessing a web-based service from a smartphone. The evaluation is based on a real-world public transport application. Travel information is a good example of information requested from users en route, and hence, an application of special interest for smartphones. Previous studies in the literature ranges from trip planning
II. WEB SERVICES

A web service is commonly used to exchange data between distributed systems over a network using standard-based Internet protocols, and is therefore, one possibility to implement a client/server interface.

The official definition of a web service was given in a note released by the W3C [7]:

"a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards. A web service is an abstract notion that must be implemented by a concrete agent."

This definition already contains a number of protocols, but does not state constraints about programming languages or operating systems. This leads to the most important advantage of web services. They can be used in different environments, using different programming languages by communicating via standardised communication protocols and data exchange formats. Figure 2 illustrates the communication between client and server using a web service.

A. Web service architecture

The following is the definitions for the terms to explain the architecture of web service from [7].

Agent: a computational resource that implements web services

Provider: a person/organisation that provides an agent to implement the service

Requester: a person/organisation that uses the provider's web service

Service description: a machine-processable specification of the web service interface (written using web Service Description Language, WSDL.)

As the entire architecture is quite complex it is preferable to divide it into several parts, where each part highlights different aspect of web services. The following paragraphs gives a short overview of the four different models, which are depicted in Figure 3, form the web service architecture, namely (see Figure 3):

- Service oriented model: This model focuses on the meta-data that describes the service, the provider that owns the web service, and the agent that realises on it. As a service relates to messages, the messages are also part of the view.

- Message oriented model: Web services are explained from the perspective of the message itself. This includes the structure of the message (body, header) the message protocol, the agent (as sender and receiver), delivery policies, etc. This model does not attempt to show the relation between messages and services.

- Resource oriented model: This model shows the relation between resources and their owners. Each resource is identified by a URI and can have several representations (e.g. the data and content-type of the resource).

- Policy model: The focus lies on constraints that are applied to agents, services or resources and have impact on whether an action is performed by the agent or not. These policies are established by the provider.

The four models give an overview of important parts of a web service in an abstract way, but they do not make a statement about technologies that are supported to realise the web services. Such technologies can be termed as web service architecture stack. This stack is
organised in several layers as depicted in Figure 5. From the bottom up, the first layer, Transport, is responsible for the transport of the messages between provider and requestor. Technologies like HTTP, SMTP or FTP can be used for transmitting the messages. The message layer is responsible for putting each message in a given format. As a common protocol, SOAP can be used. The description of the web service and its components is realised as a separate layer using WSDL. At the top, the processing layer handles functionality like discovering a web service. Discovering is defined as “the act of locating a machine-processable description of a web service that may have been previously unknown and that meets certain functional criteria” [7].

![Figure 5: Web service architecture stack adapted from [7]](image)

**B. Web Services Styles**

The web service architecture stack shown in Figure 5 already contains a number of technologies that can be used to build and access web services which offers the developer many possibilities to concretise the web service. Besides the used technologies, it is possible to define three styles for using web services, thereby each possibility brings another aspect of the architecture (see Figure 3) to the four. The three web service styles are [7]:

- Remote Procedure Call (RPC)
- Simple Object Access Protocol (SOAP)
- Representational State Transfer (REST)

**C. RPC web services**

As the name implies, RPC allows the client to call a procedure (also called method) that is executed on another system within a distributed system, thereby the client and server do not have to be aware of the underlying network. Most RPC web services use XML-RPC, a protocol that uses XML to describe the calls, which are then transmitted between the client and the server using HTTP. To simplify the use of XML-RPC, the three technologies RPC, XML and HTTP are combined so that client and server can share resources over a network. The principle is quite simple: the procedure call only requires the name of the procedure and a number of parameters, to return a result [17]. Figure 4 illustrates an RPC web service and the structure of the RPC message containing the procedure (XML-tag methodName) and parameters (XML-tag params).

Instead of XML for the description of the remote procedure call, JSON can also be used [18].

The disadvantage of RPC web services is that the abstraction level is quite low due to the simplicity of the RPC architecture. As, the complexity increases with an increasing number of different requests [17]. Due to this limitation, the simple RPC web services will not be considered further.

**D. SOAP web services**

SOAP is a protocol for exchanging information in a distributed system and relies on XML for its data format. SOAP is the successor of XML-RPC. SOAP fits in on the top of the network and transport layers and can be used in combination with any transport protocols. Due to its pervasiveness, HTTP is the most common transport protocol used to exchange SOAP messages.

The structure of a SOAP message can be seen in Figure 7. A SOAP message consists of a SOAP Envelope that contains an optional SOAP Header element and a required SOAP Body. While the optional SOAP header can be used to define how the message has to be processed (e.g. routing information, authentication assertions) the SOAP
body contains an actual valid, well-formed XML message [2].

Figure 7: SOAP message adapted from [2]

Figure 6 illustrates a SOAP web service and the structure of the SOAP message. Although the figure contains a request and a response message, SOAP messages are fundamentally one-way transmissions. Combinations of messages (such as request-response) are possible by using so-called Message Exchange Pattern [19]. When a SOAP message is sent from a sender (client) to a receiver (server) or vice versa the message may pass several intermediate nodes where it is processed and maybe changed. In that way, the SOAP headers can be used to specify which parts of the message are intended for processing by a specific intermediate node.

SOAP also allows the definition of special messages that are sent when an error occurs during the processing of the SOAP message (called SOAP Fault). SOAP Faults are sent in the SOAP Body and contain a fault code, an explanation of the error and identifier of the processing node, where the error has occurred, and application-specific details about the error [2].

When using HTTP for transmitting SOAP messages, only HTTP POST or GET methods can be used. The SOAP message has to be placed in the HTTP entity body. The HTTP headers can be used to set the content-type to the defined media type “application/soap+xml”. Additionally, the HTTP request header SOAPAction must be set, possibly, as an empty string. This means that the intent of the SOAP message is indicated by the Uniform Resource Identifier (URI). Figure 8 shows an example of SOAP using HTTP as the transport protocol [19].

Since SOAP version 1.1 was submitted to W3C as a successor of XML-RPC in 2001 [20], there has been remarkable development in this area. The XML Protocol Working Group within W3C developed the protocol further, and the results were announced in a new version SOAP 2.1, as a W3C recommendation in 2003 [21]. One of the changes was that SOAP no longer is an acronym, as the expansion of “Simple Object Access Protocol” is misleading.

E. REST web services

Fielding [22] defined the term Representational State Transfer (REST) in his dissertation as an “architectural style for distributed hypermedia systems”. The best known example following the REST style is the World Wide Web (WWW). While the web service styles, RPC and SOAP, can be classified as service-oriented web services, REST web services focus on resources, as clarified in Table I.

Table 1: REST data elements adapted from [22]

<table>
<thead>
<tr>
<th>Data Element</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource</td>
<td>intended conceptual target of</td>
</tr>
<tr>
<td>resource identifier</td>
<td>a hypertext reference</td>
</tr>
<tr>
<td>representation</td>
<td>URL, URI</td>
</tr>
<tr>
<td>representation metadata</td>
<td>JPEG image, HTML document</td>
</tr>
<tr>
<td>control data</td>
<td>source link</td>
</tr>
<tr>
<td></td>
<td>cache-control</td>
</tr>
</tbody>
</table>

Applying the REST style to web services means that each web service operation is described as a resource and can be accessed by a resource identifier. This identifier can be seen as an address which specifies where to find the resource. There are a number of terms referring to resource
identifiers like URI or Uniform Resource Locator (URL). When transferring information between components (e.g., client/server) – this means accessing a resource by a resource identifier – the state of the information received by the resource at a specific time is called representation. Next to the representation, metadata describing the media type of the representation is also transmitted [22]. Figure 9 illustrates the connection between resources, resource identifiers, representations and their metadata applied to web services.

![Figure 9: REST web service adapted from [23]](image)

F. Comparison of RPC/SOAP/REST web services

Using HTTP as the transport protocol, the difference between SOAP and REST is that SOAP uses its own envelope. This encapsulation of the actual data gives the possibility to add processing information in the SOAP header. This may increase functionality (e.g., to define which intermediate nodes are allowed to access which parts of the message), but it also increases the overhead of the SOAP Envelope when the SOAP Header is not used. In contrast, REST puts the actual data in the HTTP body without adding an envelope.

When comparing REST and RPC, the major difference is that RPC messages contain parameters and the name of the procedure. This means that it is necessary to read the message to specify which procedure to call. The procedure name is accessed by the resource identifier and is not part of the message in the HTTP body.

RPC only supports the HTTP POST method, whereas SOAP also supports the HTTP GET method. However, only REST allows the use of other HTTP methods (PUT and DELETE). This is due to the fact that REST is tied to HTTP as the transport protocol and takes advantage of more HTTP specific functions than RPC and SOAP [24].

SOAP and REST also differ in the way errors are handled. While SOAP defines SOAP Faults, that are sent in case of an error, REST uses HTTP status codes (see [25]) to specify the error.

Another important fact is that SOAP, in contrast to REST, was designed for end-to-end communication. While SOAP provides the possibility to modify the message by an intermediate node, REST assumes a point-to-point communication. SOAP and REST also differ in terms of security. REST web service securities the communication using Hypertext Transfer Protocol Secure (HTTPS) and while SOAP defines security extensions (e.g. encryption and signature) [26].

III. DATA EXCHANGE FORMATS

A. XML

XML is a text format that aims to be “usable over the internet”, “human-readable and reasonably clear” and “easy to create” [27]. XML is a markup language - this means that next to the transmitted data, an XML document also contains a description of the data. The data is described by tags (e.g., `<tag>...</tag>`) defined by the user. In terms of the structure of XML documents, each document has a physical and a logical structure. The logical structure of an XML document is composed of [27]:

- declaration
- elements
- comments
- character references
- processing instructions

If the optional declaration is used within an XML document, it must be placed at the top. The declaration gives information about the used XML version (1.0 or 1.1), the encoding, and whether an external file (called Document Type Definition, DTD) is required to be read in order to further process the document. An example for a declaration is: `<?xml version="1.0" encoding="ASCII" standalone="yes">`.

An XML document has to consist of one root element, which might have a number of nested elements, that forms a tree. Each element within an XML document is identified by a name, an element type, optional attributes, and has to be either bounded by start-tag/end-tag or uses an empty-element-tag. Inside an element (between start-tag and end-tag), the actual user data (content) can be placed as a sequence of characters (character references). Processing instructions can be used to pass information to the application which processes the XML document and are indicated by question marks `< ? ... ? >` within the tag. The physical structure of the XML document represents the storage units (called entities) and the content of the entities [28].

A simple example of an XML document where the elements of the logical structure are highlighted is shown in Figure 10.

![Figure 10: Example of XML document](image)
B. JSON

JavaScript Object Notation (JSON) is defined as “a lightweight, text-based, language-independent data interchange format” and “a set of formatting rules for the portable representation of structured data”. It was originally specified in the RFC 4627 by Douglas Crockford [29].

A JSON message is composed of the elements shown in Table II and is represented by an object or an array. An object is bounded by a pair of curly brackets indicating the beginning and the end of the object element. An object contains zero or more name/value pairs within the brackets. The name and value are separated by a colon. Several pairs are separated using a comma.

<table>
<thead>
<tr>
<th>Structured bytes</th>
<th>Primitive types</th>
<th>Literal names</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>string</td>
<td>false</td>
</tr>
<tr>
<td>array</td>
<td>number</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>null</td>
</tr>
</tbody>
</table>

An array element is bounded by a pair of square brackets indicating the beginning and the end of the array element. An array contains an ordered collection of zero or more values within the brackets, where the values are separated by a comma.

A value can be a primitive type, a structured data type (leads to nested elements), or one of the three defined literal names [29]. Figure 11 shows an example of an JSON message where the JSON elements listed in Table II are highlighted.

Figure 12 illustrates how Fast Infoset uses indices in order to reduce the amount of overhead. The Fast Infoset document within the figure only illustrates the use of indices in a human-readable way, while an actually implemented Fast Infoset document stores the information in a binary format [31].

D. Comparison XML/JSON/Fast Infoset

The most important distinction among the considered data formats is that XML and JSON are human-readable, while Fast Infoset data is represented in a binary format. It is more difficult for the developer to recognise errors within the transmitted data in distributed systems when a binary format is used. Another factor, that is important from the developer’s point of view, is the availability of parsers for processing the document represented by a specific data format. While XML is widely-used and therefore lots of parsers (e.g. Simple API for XML, SAX, Document Object Model, DOM) are available in different programming languages, it is more difficult to find an appropriate parser for processing JSON or Fast Infoset documents.

Factors, that are of importance for the end-user, are the transmitted document size and processing time of the document (serialisation/deserialisation time). Bruntaset [32] refers JSON as the “fat-free” alternative to XML, as JSON organises the data in a more compact way than XML. This statement is confirmed when comparing the syntax of JSON and XML. XML is more verbose as it requires a start-tag and end-tag for describing the content, while JSON does not use end-tags at all. JSON objects/arrays are only concluded by one single character. Hence, JSON enables the developer to represent the same data by using fewer characters than XML. Similar to JSON, Fast Infoset also only uses tags to indicate the beginning of an element. In addition, Fast Infoset replaces repeating textual content by using indices. This leads to an additional reduction of the file size.

Regarding the processing time, Fast Infoset documents can be processed faster than those of XML as entire blocks are read instead of character by character. Generally, the processing time depends on the use of the specific parser.

C. Fast Infoset

The standard Fast Infoset is a draft, that specifies a binary encoding format for XML information sets [30]. Fast Infoset documents are analogous to XML documents, but generally smaller in size. Fast Infoset documents are known to be faster to serialise and deserialise [30].

This is achieved by:
- tags are only used to indicate the beginning of an element.
- the length of each data block is stored to enable the parser to read entire blocks instead of character by character.
- the repeating of textual content and qualified names can be avoided by using index tables.

IV. SERIALISATION/DESERIALISATION ON THE SMARTPHONE

Serialisation/deserialisation is a crucial part of the application and the performance of this task is mainly subjected to its implementation. A short overview of the used frameworks and adaptations for performing this task follows. As the implementation of serialisation/deserialisation depends on which data format is used, the description is divided into serialisation/deserialisation of

- JSON message
- XML message
A. Serialisation/Deserialisation of JSON message

Currently, there are only two libraries available for serialising/deserialising of JSON messages. These third-party libraries are JSON-Framework [33] and TouchJSON [34]. Both of these libraries support the serialisation of the following Objective-C objects into a JSON string and vice versa (deserialisation):

- NSDictionary

Based on the data structure defined in the WSDL, NSDictionary is the only suitable class supported by the libraries for representing the data. As it is more structured and preferable for custom Objective-C objects, which are based on the defined WSDL structure, it was necessary to implement a method for “transferring” the resulting NSDictionary (after serialisation using the library) into the custom Objective-C objects. As NSDictionary does not store data in an ordered way, while the GTI web service expects data in a specific order (required due to the use of JAXB-B), a workaround is necessary. An “ordered NSDictionary” was implemented as a workaround by subclassing the NSMutableDictionary class and keeping the order in which the elements are added to the dictionary. This approach was implemented by using the third-party library TouchJSON.

B. Serialisation/Deserialisation of XML message

In comparison to JSON, there are several third party libraries available for serialising/deserialising XML messages. However, most of them (including the Objective-C library within the iPhone SDK NSXMLParser [35]) only support methods for reading and parsing XML messages into custom objects (deserialising). Libraries that support both serialisation and deserialisation are:

- KissXML [36]
- GiDataXML [37]

Both libraries are based on libxml (an XML parser written in C) and provide the functionality to read and write XML data. GiDataXML is a part of Google Data APIs Objective-C Client Library, which provides a number of functions required to create an SmartPhone client to access Google services. KissXML, on the other hand, only contains the necessary functionality to handle XML data. Since only a fractional amount of the Google Data APIs Objective-C Client Library would be needed, and due to lack of documentation, the KissXML library was used.

KissXML is a DOM style XML parser that provides the classes DDXMLODocument, DDXMLElement and DDXMLONode for parsing XML messages into custom Objective-C objects and to create XML messages based on custom objects. KissXML also supports the use of namespaces within the XML message. The serialisation is performed by creating DDXMLElement objects corresponding to the data stored in the Objective-C object and lined up in the required order. The deserialisation is performed by iterating through all DDXMLElements within the DDXMLODocument and transferring the data into custom Objective-C objects.

V. IMPLEMENTATION

This paper represents an overall general performance evaluation of a real life application of different web services. Clearly, there are several factors, which do influence the empirical results. It is beyond the scope of this presentation to consider, for example; web service performance greatly depends on the programming skills of a developer and technology adopted; using different technologies may cause different performance results; there are different algorithms to implement the same functionality, which would also cause big differences in performance results.

For the comparison, an existing SOAP web service which provides public transport information for the city of Hamburg has been used. In order to enable a comparison with REST style web services, a new REST web service (called GTI web service) has been implemented, which provides the same functionality as the existing SOAP web service. Three existing features, namely init (get general information), checkname (request existing start/destination points to search for a route) and getroute (request an
optimised route from a start to a destination point) were used. Both web services are accessed by HTTP.

There are several vendors for smartphones on the market. For practical reason, we have limited our experiments to one of them, Apple’s iPhone. The test were done on an iPhone 3G-S, 8GB flash drive, connectivity for UMTS/HSDPA (850, 1900, 2100 MHz), GSM/EDGE (850, 900, 1800, 1900 MHz), 256 MB DRAM Memory on a 3.5-inch (diagonal) widescreen Multi-Touch display 480-by-320-pixel resolution at 163 ppi Fingerprint-resistant oleophobic coating. They were performed over a normal working day. A native iPhone application has been implemented that accessed both web services.

Since the data used in the HTTP request, determines how much information is returned from the web service response, it is important what kind of data is used in the HTTP request. In our implementation, we define two types of messages, one simple and one detailed:

- simple message
  - simple request: A request that provides the web service method with information and expects a simple response.
  - simple response: A reply to a simple request that contains a minimum of results from a specific web service method
    * init: general information, no optional properties
    * checkname: one start/destination point
    * getroute: one optimal route, containing one trip part

- detailed message
  - detailed request: A request that provides the web service method with information and expects a detailed response.
  - detailed response: A reply to a detailed request that contains several results from a specific web service method
    * init: general information, a number of additional properties
    * checkname: all start/destination points that match the requested information
    * getroute: one optimal route, containing several trip parts

The use of these two message types provides the possibility to determine the range of delay times that an end-user can expect. Table III exemplifies the simple and detailed message types for the init web service method.

In order to measure the performance indicators, it was necessary to place measurement points. All these were placed within the iPhone application to avoid synchronisation errors between different platforms. Based on the performance indicators, the iPhone application was used to measure:

- transferred data volume
  - content length of HTTP request
  - content length of HTTP response
- elapsed time during

- serialisation process
- deserialisation process
- the execution of the HTTP request and the reception of the HTTP response

In this context, the transferred data volume is defined as the number of bytes of the message that is placed in the HTTP body of the request and response, respectively, using UTF-8 encoding.

The calculation of the elapsed time is performed using NSDate to represent a single point in time and is immutable. An object that represents the current date and time was created using the method date. For the determination of the elapsed time, the instance method timeIntervalSinceNow is used which returns the time interval between the calling of the NSDate object and the current date and time. The return type is NSTimeInterval which specifies a time interval in seconds [38]. The return type NSTimeInterval refers to a double precision, floating-point value, and yields sub-millisecond precision. The floating-point value enables a calculation of the time difference in milliseconds.

For computing the serialisation/deserialisation time, measurement points were placed right before and after the invocation of the respective method from the third party library.

Figure 13 illustrates the entire serialisation and deserialisation process, that is performed when a web service method is invoked by the iPhone test application.

The results were stored in an appropriate format after the measurements were conducted on the iPhone in order to perform further analysis. For this purpose, a separate performance test web service was created that receives the measurement results, which were gathered using the iPhone application. The mobile user was fixed when sending and receiving the data.
VI. Results

This section presents and illustrates the measurement results of the performance indicators:

A) transferred data volume [bytes]
B) request–response time [ms]
C) serialisation/deserialisation time [ms]

The result table for transferred data volume (Table IV) shows the message size of the different message types (simple/detailed) that were sent and received from the various web service methods (init, checkname and getroute) during the performance test. For the performance indicators request–response time, serialisation time, and deserialisation time, the means and the standard deviations, calculated from a sample size of 30 measurements, are presented in Table V, VI and VII, respectively.

The gathered results enable a comparison of the SOAP web service against a REST web service. It is also possible to compare the different data exchange formats that have been used. The corresponding results for the existing SOAP web services are included in the overview tables (Tables IV, V, VI, VII).

A. Transferred data volume

Table IV illustrates the data volume transferred when the web service methods of the REST web service using the data formats, JSON, XML, Fast Inset and the SOAP web services, are accessed with either the simple or detailed message types. As there are no libraries for the serialisation/deserialisation of Fast Inset messages available for the iPhone, the data volume of Fast Inset was determined implicitly by converting the XML message that was transmitted to the web service to a Fast Inset message.

The table shows that the used data format (XML, JSON or Fast Inset) has a significant impact on the number of bytes that are transferred. When the XML data format is used (in the case of REST XML and SOAP) the highest amount of bytes are transferred. This is the case for the requests and responses of both simple and detailed messages. It was observed that the use of the data formats JSON or Fast Inset leads to a reduction in the number of bytes that are transferred. While the use of JSON results in the smallest number of bytes for simple messages, Fast Inset shows the smallest number of bytes for detailed messages.

Comparing the new REST XML web service to the old SOAP based web service, a reduction of the amount of data of at least one third was observed.

B. Request–response time

Table V illustrates the average and standard deviation (request–response) times when the web service methods of the REST and the SOAP are accessed using different network connections on the iPhone. The values of the request–response time using a 3G network connection are lower than that of 2G networks, because the 3G network connection provides higher data rates. For a complete analysis of the 2G results, please refer to [39].

There are small differences between the two REST methods, but both are remarkably faster than the old SOAP method.

C. Serialisation/Deserialisation time

Table VI summarises the elapsed time during the serialisation process that was performed in order to create
the HTTP request to access the SOAP and the REST web service methods, using the data formats JSON and XML. Fast InfoSet turns data into a binary format. This is an efficient way of sending data, as there is no, or very little, overhead involved. The downside is that due to the missing metadata, the message can only be deserialised if the exact serialisation method is known. Even if the device knows the exact serialisation method, there is considerable overhead changing binary format into serialisation format when working with devices that have limited computational power, memory, and low speed data connections like a mobile device [40]. As a result, there are no libraries for the serialisation/deserialisation for Fast InfoSet. As the request messages of simple and detailed type do not differ significantly in the message size, the measured serialisation times of simple and detailed request messages are consequently in a similar range. Further, Table VI shows that the serialisation of messages represented in XML messages requires less time than the serialisation of the same messages represented in JSON. Furthermore, messages represented in REST web service methods (JSON and XML) are serialised faster than those in SOAP.

Table VII summarises the time that is required for the deserialisation of the HTTP response received from

| request–response time [ms] when using iPhone network connection |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
|                           |  simple message type | detailed message type |  simple message type | detailed message type |
|                           | average              | standard deviation | average              | standard deviation |
| REST                       | 344.83              | 49.47              | 471.86              | 122.07              |
| JSON                       | 315.69              | 37.55              | 426.96              | 94.25               |
| REST                       | 325.75              | 51.33              | 549.92              | 61.99               |
| XML                        | 344.83              | 79.55              | 662.89              | 90.29               |
| SOAP                       | 358.56              | 96.32              | 843.33              | 63.38               |
| getroute                   | 355.43              | 63.32              | 3314.09             | 420.82              |
| REST                       | 308.57              | 41.39              | 5068.43             | 465.75              |
| XML                        | 1870.33             | 277.38             | 5559.70             | 408.21              |

Table VII: Results: deserialisation time [ms] |

| deserialisation time [ms] |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
|                           |  simple message type | detailed message type |  simple message type | detailed message type |
|                           | average              | standard deviation | average              | standard deviation |
| inits                      | 2.02                | 0.03               | 3.87                | 0.10               |
| REST                       | 2.89                | 0.04               | 4.45                | 0.05               |
| JSON                       | 2.42                | 0.16               | 5.50                | 0.20               |
| REST                       | 2.55                | 0.04               | 11.36               | 0.41               |
| XML                        | 4.18                | 0.06               | 51.32               | 0.46               |
| SOAP                       | 5.93                | 0.13               | 40.68               | 0.59               |
| getroute                   | 16.17               | 0.48               | 101.50              | 2.23               |
| REST                       | 25.59               | 3.39               | 93.81               | 9.84               |

The REST web services with XML was compared to the data format with the SOAP web service, the use of the REST web services showed a reduction in serialisation time of about 4/5 as a whole for all message types.

Table VIII: Relative reduction of the transferred data volumes when changing SOAP to REST |

| relative reduction of the transferred data volumes when changing SOAP to REST |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
|                           |  simple message type | detailed message type |  simple message type | detailed message type |
|                           | request              | response             | request              | response             |
| inits                      | 0.96                | 0.93                | 0.98                | 0.95                |
| REST                       | 0.90                | 0.83                | 0.97                | 0.93                |
| JSON                       | 0.91                | 0.87                | 0.98                | 0.95                |
| REST                       | 0.95                | 0.95                | 0.85                | 0.80                |
| XML                        | 0.89                | 0.89                | 0.73                | 0.69                |
| REST                       | 0.92                | 0.91                | 0.97                | 0.95                |
| Fast InfoSet               | 0.91                | 0.91                | 0.97                | 0.95                |
| REST                       | 0.88                | 0.88                | 0.97                | 0.93                |
| XML                        | 0.92                | 0.92                | 0.98                | 0.98                |

Table IX: Relative reduction of reduction of the request–response time by changing SOAP to REST |

| relative reduction of reduction of the request–response time by changing SOAP to REST |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
|                           |  simple message type | detailed message type |  simple message type | detailed message type |
|                           | request              | response             | request              | response             |
| inits                      | 0.40                | 0.11                | 0.45                | 0.19                |
| REST                       | 0.48                | 0.18                | 0.66                | 0.42                |
| JSON                       | 0.60                | 0.20                | 0.64                | 0.30                |
| REST                       | 0.01                | 0.14                | 0.44                | 0.25                |
| XML                        | 0.20                | 0.45                | 0.30                | 0.45                |
TABLE X: Relative reduction of the serialisation time when changing SOAP to REST

<table>
<thead>
<tr>
<th>init</th>
<th>simple message type</th>
<th>detailed message type</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST JSON</td>
<td>0.55</td>
<td>0.33</td>
</tr>
<tr>
<td>REST XML</td>
<td>0.81</td>
<td>0.72</td>
</tr>
<tr>
<td>REST JSON</td>
<td>0.38</td>
<td>0.28</td>
</tr>
<tr>
<td>REST XML</td>
<td>0.91</td>
<td>0.80</td>
</tr>
<tr>
<td>REST JSON</td>
<td>0.20</td>
<td>0.19</td>
</tr>
<tr>
<td>REST XML</td>
<td>0.72</td>
<td>0.72</td>
</tr>
</tbody>
</table>

TABLE XI: Relative reduction of the deserialisation time when changing SOAP to REST

<table>
<thead>
<tr>
<th>init</th>
<th>simple message type</th>
<th>detailed message type</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST JSON</td>
<td>0.73</td>
<td>0.59</td>
</tr>
<tr>
<td>REST XML</td>
<td>0.64</td>
<td>0.53</td>
</tr>
<tr>
<td>REST JSON</td>
<td>0.39</td>
<td>0.26</td>
</tr>
<tr>
<td>REST XML</td>
<td>0.52</td>
<td>0.47</td>
</tr>
<tr>
<td>REST JSON</td>
<td>0.80</td>
<td>0.72</td>
</tr>
<tr>
<td>REST XML</td>
<td>0.43</td>
<td>0.38</td>
</tr>
</tbody>
</table>

D. Potential of improvement over SOAP

The overall result of the study presented above is, that there lies a great potential of improvement in replacing the old SOAP web service with REST. Using the existing SOAP web service as a benchmark, we can study the degree of improvement in the system performance by changing to REST in some data format. Table VIII, IX, X and XI show the potential reduction of transferred data volume, request–response time, serialisation time, and deserialisation time, respectively.

The reduction of request–response time (Table IX) is significant, both for the JSON and the XML data formats, with the least improvement (11%) reported for the web service method init (detailed message type) using the JSON data format. Best reduction of request–response time, 81% compared to the old SOAP web service, is achieved for the method getroute (simple message type) with the XML data format. Compared to getroute, however, XML and JSON are relatively close, since SOAP has a much longer request–response time.

The potential improvement in serialisation time is, as reported in Table X, also large; up to more than 80% and never below 30%, which is achieved with the web service method getroute using the JSON data format (both method types).

E. Discussion

A comparison of the different data formats of JSON, XML, and Fast Infoset shows that the use of the XML data format results in the highest amount of transferred data (see Table IV). This is due to the fact that XML is characterised by its rich verbosity, meaning that it requires a separate start-tag and end-tag for describing the content. As JSON does not use end-tags at all for the description of the content, the resulting number of bytes is smaller. Fast Infoset shows a similar number of transferred bytes as does JSON for simple messages. As for JSON, the explanation lies in the elimination of end-tags, which in Fast Infoset, are replaced by index tables, where all elements that occur within the document are stored. This approach is inefficient for messages that mostly consist of unique elements, but beneficial if a reproduction of existing content can be avoided by repeating given indices.

In the considered application, the request messages usually contain only a small amount of data, and hence the use of Fast Infoset, generally, does not reduce the number of transferred bytes. The only messages that contain enough redundant textual content for a significant reduction of transferred bytes are the detailed response messages of the getroute web service method (see Table IV), which comprises of several trip parts that use the same redundant start- and end-tags. For request messages and simple response messages the use of JSON results in the lowest number of transferred bytes. It can be concluded that the use of the XML data format is not efficient in terms of transferred data volumes.

Besides, the use of the JSON, XML and Fast Infoset web services leads to a significant overall reduction in the number of transferred data volume in comparison to the use of the old SOAP web service (see Table IV and

TABLE XII: Summary performance test: best performing web service/data exchange format

<table>
<thead>
<tr>
<th>transferred data volume</th>
<th>simple message type</th>
<th>detailed message type</th>
</tr>
</thead>
<tbody>
<tr>
<td>init checkname getroute</td>
<td>JSON Fast Infoset</td>
<td>JSON Fast Infoset</td>
</tr>
<tr>
<td>serialisation time²</td>
<td>init checkname getroute</td>
<td>XML - XML - XML -</td>
</tr>
<tr>
<td>deserialisation time³</td>
<td>init checkname getroute</td>
<td>- JSON - JSON -</td>
</tr>
<tr>
<td>request-response time³</td>
<td>init checkname getroute</td>
<td>XML JSON/JSON/JSON/JSON/</td>
</tr>
</tbody>
</table>

¹Fast Infoset was not considered due to lack of Fast Infoset support on the iPhone

© 2012 ACADEMY PUBLISHER
VIII checkname and getroute) which is not designed for smartphones at all. The reason is that, in the considered application, the use of SOAP introduces a constant overhead of 261 bytes that is needed for the SOAP Envelope and SOAP Body elements. Additional overhead occurs due to the fact that the information sent from the SOAP web service methods is not adjusted to the requirements of smartphone clients. This means that, for instance, the getroute web service method of the SOAP web service returns three departure times for each route, although it is not possible to display this information on one single screen on the smartphone.

As the test application has been developed for iPhone, it should be noted that the measurement results of the serialisation and deserialisation processes shown in Table VI, VII, X and XI depend on the used iPhone libraries, namely TouchJSON and KissXML. It is not possible to draw conclusions on the serialisation/deserialisation times for other smartphones.

The results of the performance test show that the serialisation process of XML is faster than that of JSON for all tested web service methods and message types. The serialisation into an XML message can, unlike JSON, be directly performed from the Objective-C object, which explains the first serialisation results for XML, given in Table VI. The much longer serialisation time needed for the SOAP web service is due to its design and overhead. Also, for the deserialisation, the REST web service methods were faster than the SOAP web service, but the differences were smaller. The structure of the SOAP web service, with a demanding overhead, makes it relevant only for the most detailed message types. These are, however, not relevant for a smartphone application, since the amount of information, that can be shown at a time, is limited by the screen size.

VII. CONCLUSION

Table XII summarises the results of the performance tests and contains the web services and data exchange formats that show the best results for a specific performance indicator. The best results for every single performance indicator were achieved by using the new implemented REST web service. The data exchange format (JSON, XML or Fast Infoset) that shows the best results depends on the performance indicators and message types. In some cases it was not possible to determine significant differences between the use of JSON and XML.

The overall results of the performance tests have proven that the use of the REST web service leads to an improvement in terms of transferred data, serialisation/deserialisation time and request–response time (in contrast) to the old SOAP web service. Which data format is the most efficient depends on the mobile platform for which the native client application is developed. As the web service is not tailor-made to a specific mobile platform it provides the support of JSON, XML or Fast Infoset. For the development of native iPhone applications, the use of the data exchange format JSON is recommended. Although it was not possible to measure the request–response time and processing time of Fast Infoset messages, the comparison of the transferred data volumes has shown that the use of Fast Infoset is quite promising for the future when more libraries for Fast Infoset are available.

The tests in this paper are limited to one single application. We believe that this application is a typical example for a public transport information system, which is an area of growing interest for smartphone applications that can enable travel information en route. To strengthen the conclusions, however, we would welcome more tests, also from other application areas.

ACKNOWLEDGEMENTS

The presentation of the paper has benefited from the comments of two anonymous reviewers, to whom the authors express their gratitude.

REFERENCES


