UNIVERSITY OF GOTHENBURG

Performance & Implementation Best Practices for Information Passing Through Web Services

XIAOMING CAI <ming.cxm@gmail.com>
JOSEFINE OTTOSSON <josefine.ottosson@gmail.com>

Bachelor of Software Engineering & Management Thesis

Report No. 2009:050
ISSN: 1651-4769

University of Gothenburg
Department of Applied Information Technology
Gothenburg, Sweden, May 2009

Abstract - *Data interchange between different web applications and web-services becomes more and more common. Not only to send or receive data from another service or application but also to reuse modules and web-service functionality from one application to another. There are many standards and protocols available today to use when building web-services. This research will only focus on three of them which all represent some of the most common implementations when building web-services of today. These are XML-RPC, SOAP and REST. They have differences between each other but can perform the same tasks for a web-service. They are examined through performance and implementation issues, benchmarking as well as interviews. The result reveals performance issues for XML-RPC as well as how complex SOAP can be for inexperienced users. In addition it also reveals why REST is gaining popularity recently.*

1. Introduction

The purpose of this paper is to present the best practices for information passing through web-services. Information passing in this context means sending requests and answers between a client and a server. It can be done through different ways of implementation through standards and/or architectures. This paper will examine three of the most common ways, which are SOAP [1], XML-RPC [2] and REST [3]. These will be referred to "research items" in the following paper. SOAP and XML-RPC are standards for transferring data through web-services. Both of them have interfaces to the client and server and runs above a transport protocol. REST on the other hand is more of an architecture as defined by Tomas Fielding (1). It defines a new way of representing data in various representations.

[1] Simple Object Access Protocol
[2] Extensible Markup Language – Remote Procedure Call
[3] Representational State Transfer

When choosing a standard and architecture to use for a web-service, a lot of aspects have to be taken into considerations since the various standards differ from each other. For example, for a performance requirement, bandwidth and memory usage, one standard could be better than others to use. Time pressure for the development could be crucial as well. Therefore, choosing a suitable standard to implement is beneficial.

In this research, we have looked at performance issues as well as implementation issues for these three research items. The performance issues examined are network latency, XML parsing latency and query latency. The implementation issues include, for example, customization, data types, transport protocol, multiple request handling, data overhead and extensibility/ maintainability.

In order to measure performance and implementations issues, a benchmarking has been performed together with interviews. Through those means, this study has collected enough data to let readers have a clear overview of all research items from different perspectives. In the end, this study presents a model for how to use the research items and in what situation. When choosing what standard or architecture to use for a web-service, the model can help clarify the differences between the various research items.

The paper is structured in the following manner: Next section is the literature review which presents other research in the field and statements about the three research items. The literature review is followed by the Research Approach section where the research method is explained together with the grounded theory. After that the Result section presents results from both the benchmarking and the interviews. The Discussion section presents a summary of the results and the emerged model. Last is the Conclusion that sums up the whole paper and presents future work.

2. Literature Review

The literature review will reveal implementation and performance issues which have been noted by other researchers.

SOAP is widely used as a communication protocol for data exchanging in XML-format (2). Implementation-wise it is said to provide simplicity, robustness, extensibility and interoperability to a web-service (2) (3) (4). It has good support for implementation customization by supporting, for example customized types (5) and it is not bound to any transport protocol (6).

On the other hand SOAP's usefulness is threatened by its poor performance (2) (4). According to Nayef et al. (2) and Ng et al. (7), it is the high overhead due to serialization to the outgoing XML message that is the big bottle neck.

SOAP is described through the WSDL [4] document, which is an XML-interface specification between the client and the server (4). WSDL has become a de facto standard to use together with SOAP (4). However, it has been said that for non-programmers, SOAP with its WSDL document can be hard to overview (8).

REST is said to be inexpensive, simple and easier to extend than XML-RPC implementations can offer (9). It only elaborates the essential parts for internet-based hypermedia interactions. As an architecture, it keeps its abstraction rather than details of implementation where the details can be replaced if necessary (10). REST has also been used to explain the excellence and scalability of HTTP. Thus, it is commonly used in conjunction with HTTP (11). However, REST is criticized as "lacks tooling and interface definition languages, or that it works for human driven browser-based systems but is unsuitable for application-to-application integration, and it can't adequately support distributed transactions." (9) p. 94.

Implementation of XML-RPC into an application is said to be easy and to reduce the programming complexity a lot (12). It can easily be layered on top of existing application protocols, for example HTTP (12). It also represents loose coupling between hosts (12). But it has been extensively criticized for its network performance (13) (7) (12) (14).

3. Research Approach

During this research, we have focused on two broad aspects which may be quite important when considering choosing a research item to use. Benchmarking may help people discover how all research items perform under different conditions whereas implementation issues collected by interviews also cover other relevant parts such as some non functional requirements. Therefore, both quantitative and qualitative data needed to be collected and analyzed.

3.1 Research methodology

The grounded theory method was used for this research, one of the main reasons for choosing this method was because of its support for both qualitative and quantitative data. It is a qualitative research method which also supports collecting of quantitative data. In grounded theory, collection of data is done first and from the observations a theory emerges, which also makes it a good choice for this research. To support our statements both from the benchmarking and the interviews we highly use related literature and research. This was another reason to choose grounded theory, because of its support from existing literature and research.

The following sections describe the two parts of data collection and how it is applied to grounded theory.

[4] http://www.w3.org/TR/wsdl

Benchmarking

The benchmarking was done through two different implementations. First the benchmarking was built in PHP with built-in functions for SOAP and XML-RPC and because the version of PHP we used has no support for REST, we built it ourselves. Secondly, we also implemented the benchmarking in Zend Framework [5]. Frameworks are commonly used nowadays and have great support for all of the research items.

Both of these implementations followed the structure shown in Figure 1. One client and one server were built for each research item. The query latency was measured starting from when the request was sent and ending when the client receives the response, see Figure 1. Network latency is a big factor when running on two different hosts, therefore we also measured the time for the server when just answering with a string. Another factor for latency is the XML-parsing therefore we also did a small XML-parsing latency test.

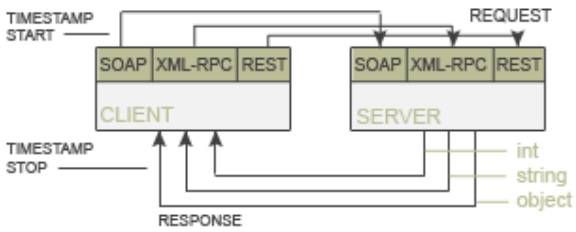

Figure 1 - Benchmarking structure

PHP was used as the application language of the benchmarking, mainly because PHP has shown to have both good performance and productivity (8) (15) (16). PHP's good performance result is based in its fast XML parser, libxml2, written in C (8) (17). A general view on web-service performance is that it is the XML-parsing and formatting that is the largest factor of latency no matter what language is used (4) (8).

[5] http://framework.zend.com/

The client and server were hosted on two different Debian hosts. The servers run on an Intel Pentium Dual CPU E2220 @ 2.40GHz with Debian etch and 4GB ram. The PHP version is 5.2.0, the clients are built on a Debian machine with the PHP version 5.2.9.

Interviews

The interviews were conducted with developers familiar with web development and mainly web-services. Both regular interviews and email interviews were performed during the study. The interviews revealed implementation issues for each of the research items. The questions asked during the interviews covered topics like what are the advantages/disadvantages for each research item, how much knowledge is needed, has the research item good and easy to find documentation and such. Details of the interview questions are added as 'A. Interview Questions' and 'B. Interview Data' in Appendix.

3.2 Data analysis

Our view on grounded theory is that it analyzes data in an iterative process where the data first is thoroughly analyzed word by word to find codes in the text. The interviews therefore needed to be recorded and transcribed. The codes then emerged into concepts which were grouped themes of them. By comparing each concept broader categories was formed and these categories were used as statements for each of the research items. This kind of analyze was used for each research item separately. The model was later based on the statements for each research item.

3.3 Validity Evaluation

There are some items that may affect and threat the correctness and validity of the result in this research. Therefore in the following paragraphs, analyses have been done on some items.

Test Environment

To be able to have a correct result, fair test environment is necessary. We used PHP as the

only language for test developing and also used one framework so that all the parts that we cannot control are identical. Besides, for the parts that we can control, i.e. our own implementation, they are written in very similar fashions and with same core method calls. However, there are still other issues like network latency over two different public servers that may make incorrectness of the result to a certain extent.

Research method

The codes to concepts to categories method has been criticized as a bit complicated for inexperienced researcher (18). Berg says that some of the major obstacles are to identify good codes and understand the intended meaning of a sentence (18). The coding approach has also been said to be a very time consuming method (19). Both of these assertions are probably true but we believe this research gained from the method since it gave a good ground for the statements we did for each research item and its concepts.

4. Result

The data collected is divided into two areas which are benchmarking and interviews.

4.1 Benchmarking

The result of benchmarking are divided into three sections: network latency, parsing latency and query latency. The following sections present the result from each of them.

Network latency

To see the network latency a request was made with a hardcoded string as return message. The following table shows the result.

Research Item	Latency
SOAP	0.0554907917976
REST	0.0721788525581
XML-RPC	0.0542722392082

Table 1 - Network latency

XML-parsing latency

For the parsing test, a number in *String* was parsed from the xml document to an *Int* and the latency for the parsing was measured to ensure how much it interfered with the query latency. This was done by the built in parsing function in PHP, `simplexml_load_file()`.

	200 Strings	400 Strings	800 Strings
parsing	0.00014	0.00034	0.00072

Table 2 - XML parsing latency

Query latency

The performance benchmarking for each research item was done with a request executed from a client and the return value from the server was an XML containing an array with all requested values. There were three data types tested as request return values, which were *Int*, *String* and *Object*. Besides, to be able to test the performance under different amount of request data constrains, we increased the amount of return data gradually, started with 200 items and ended with 3200. To get a fair result for each request, tests were performed 100 times for each data type so that an average of consumed time was calculated afterwards.

PHP's current stable version has no native support for REST and when implementing it in PHP without using any framework, it resulted in extremely high latency. Figure 2 shows the comparison with SOAP. REST is above 2.5 seconds when SOAP instead is below 0.5 seconds for each query. We also found that the library that supports XML-RPC in PHP has to be enabled in the hosts. This was not the case for any of the servers that we have access to.

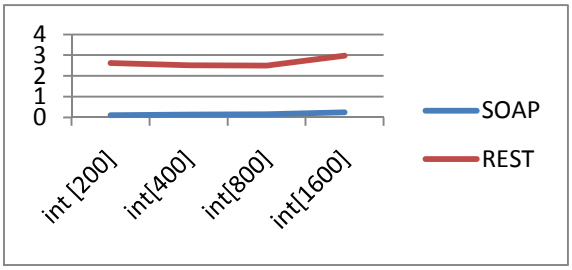

Figure 2 – First benchmarking, latency for SOAP and REST built in PHP

The following tables 3, 4, 5 show the second result of the benchmarking implemented in Zend Framework.

	Int [200]	Int [400]	Int [800]	Int [1600]	Int [3200]
SOAP	0.1970	0.2058	0.2889	0.2986	1.0335
REST	0.1600	0.2071	0.2372	0.3240	0.7548
XML-RPC	0.1819	0.1938	0.2659	0.4046	2.2575

Table 3 – Second Benchmarking, Query latency with Int

	String [200]	String [400]	String [800]	String [1600]	String [3200]
SOAP	0.1946	0.2327	0.2693	0.3562	0.8052
REST	0.2198	0.2260	0.2679	0.3679	0.8268
XML-RPC	0.2177	0.2720	0.3678	0.5563	3.6848

Table 4 – Second Benchmarking, Query latency with String

	Object [200]	Object [400]	Object [800]	Object [1600]	Object [3200]
SOAP	0.2288	0.2798	0.3669	0.5092	1.0590
REST	0.2196	0.2716	0.3435	0.4845	1.0553
XML-RPC	0.3845	0.5714	0.9083	1.6053	8.1609

Table 5 - Second Benchmarking, Query latency with Object

REST is the most stable and fastest one among all three. Both XML-RPC and SOAP are fast with arrays containing less than 1600 objects but with the increment requesting data amount, the latency started to differ, especially when requested items are above 1600, XML-RPC is suffering a lot. For objects, table 5, XML-RPC performs badly even with arrays containing few objects. Figure 2 shows an average from all three tables 3, 4 and 5.

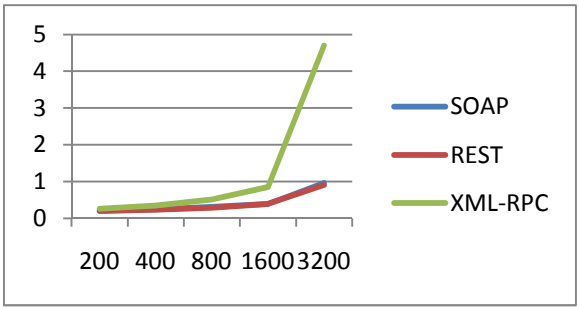

Figure 3 - Query latency average

4.2 Interviews

Data collected from the interviews was analyzed with the intention to follow the grounded theory method, from text to codes to concepts to categories. The following sections show the emerged theory for each research item from the interviews and table 6, 7, and 8 show the categories. The emerged theory will focus on three things for each research item; if it is easy or hard to use and learn, advantages and disadvantages.

Through the tables, it is possible to track the category back to its origin. The bullet points are concepts and the numbers refer to id's that could be found in each interview.

SOAP

The analysis shows that SOAP can be hard to use due to its WSDL specification. On the other hand the WSDL specification makes the SOAP implementation clean and correct.

One advantage of SOAP is that it requires strict typing. The strict typing gives correctness and a fault tolerance to the application but makes it harder to follow. The disadvantage for SOAP is the WSDL document which makes the SOAP implementation complex. One disadvantage to note is that SOAP is standardized and has a specification [6], the specification is said to change a lot and that could cause problems.

Categories	Id
THE WSDL SPECIFICATION MAKES SOAP COMPLEX	

[6] http://www.w3.org/TR/soap/

• WSDL specification is complex but necessary • SOAP is unnecessary complex with a steep learning curve • WSDL makes it time consuming and hard to maintain • WSDL takes time, knowledge and research is needed	100, 103, 101, 102, 104, 108, 114, 115, 117, 120, 121
STRICT TYPING IS GOOD BUT OFTEN UNNECESSARY	
• strict typing makes it correct and fault tolerant • strict typing is unnecessary	105, 106, 110, 111, 113, 107
THE SOAP SPECIFICATION CAN CAUSE PROBLEMS	
• the SOAP specification often updates which can cause problems • Bad specification	109, 112, 130
NO OFFICIAL DOCUMENTATION IS NEEDED	
• Easy to find information online	122
THE WSDL MAKES SOAP CLEAN AND CORRECT	
• WSDL specification is commonly used and makes SOAP clean and easy • strict typing makes it correct and fault tolerant	118, 119, 105, 106, 110, 111, 113
NEED A LOT OF PRIOR KNOWLEDGE	
• It's a nightmare and you need a lot of knowledge	123, 125, 126, 128
THE SAME LANGUAGEE SHOULD BE USED	
• Same language should be used for both client and server	124, 127, 129
BUILT-IN CLIENTS AND SERVERS EXIST BUT MAKES IT HARDER	
• A lot of built-in clients and servers • Many poor implementations exist	130, 132
LACKS IN MAINTAINABILITY	
• WSDL makes it time consuming and hard to maintain • lacks maintainability	115, 117, 133

Table 6 – Interview Data SOAP

XML-RPC

XML-RPC is commonly accepted as easy to use, it is easy to find information through tutorials and forums online since it is widely spread.

One advantage is that it is not as strict as SOAP and therefore best for open and easy services online. The disadvantages of XML-RPC are first that it is said to be closely coupled and also bad when designing an application from scratch.

Categories	Id
EASY TO USE	
• easy compared to SOAP with simple structure and no strict types • XML and some RPC knowledge is needed • easy to find other projects • framework is useful • easy to use and learn	200, 205, 206, 202, 203, 201, 204, 213, 214, 216
GOOD FOR OPEN SERVICES	
• good for loose web applications • better for open services than closed since the client does not break	207, 208, 209, 211
NO OFFICIAL DOCUMENTATION IS NEEDED	
• easy to find other projects • no official documentation • no official documentation	201, 212, 215
XML-RPC IS SIMILAR	
• XML-RPC and REST are similar	210
CLOSELY COUPLED	
• closely coupled	218
BASIC BUT FUNCTIONAL	
• basic but functional	222
WORKS AS A WRAPPER	
• wrapper around existing functions	217, 220
BAD WHEN DESIGNING FROM SCRATCH	
• not good if designing a service from scratch	221
GOOD FOR INTERNAL SYSTEMS	
• good for internal systems	219

Table 7 – Interview Data XML-RPC

REST

According to the concepts, REST seems to have a good reputation for being easily implemented. Not much prior knowledge is needed for implementing a REST service. A REST client is really simple to use and understand. In addition, a REST service is also suitable for open service as everyone can access. A common understanding would be REST is bonded to use HTTP protocol.

Categories	Id
THE CLIENT IS EASY TO USE	
• the client is easy with just an URL to manage and browser compatible • fast to implement • the client is easy to use • the client is easy and fast • easy to document and understand • standardized and makes sense • easy to use and learn	300, 301, 302, 303, 304, 310 318, 320, 322, 323, 308, 309 327, 330, 338, 339, 342, 346
USE OF HTTP AND IS GOOD FOR OPEN SERVICES	
• HTTP as transport protocol • not complex and good with open services • the complexity is unnecessary	305, 306, 307, 312, 314, 319
NO OFFICIAL DOCUMENTATION IS NEEDED	
• the documentation is application specific • used only Wikipedia and Google • knowledge about the language you use • no documentation needed • good documentation	317, 325, 326 324, 336, 340, 341
THE SERVER IS HARD TO IMPLEMENT	
• the server is complicated and takes more time • time consuming	328, 329, 334, 335
CAN'T HANDLE MANY REQUESTS	
• not good with many request • the size of data is important when choosing standard	311, 315
IT IS EASY TO MODIFY AND EXTEND DUE TO ITS FLEXIBILITY	
• it is simply to modify and extend due to its flexibility	337
EASY TO MISUNDERSTAND	
• easy to misunderstand	343
SECURE	
• secure	344
GOOD WITH VARIOUS LANGUAGES AND CLIENTS	
• good when mixing various languages and clients	345
GOOD PERFORMANCE	
• good performance	347

Table 8 – Interview Data REST

5. Discussion

Based on our qualitative and quantitative test results from previous sections, we in this section summarize and present the overall comparisons together with our own experience when conducting this research. Different aspects are discussed based on the model that we present. They are categorized into three different views: performance, technology and usability.

5.1 Performance

According to Figure 2, XML-RPC shows really bad performance above 1600 objects. The bad performance for complex and big messages is also supported in (13) (7) (12) (14).

Mentioned by Abu-Ghazaleh et al. (2) and Ng et al. (7) SOAP has high overhead and the serialization/deserialization is a big bottleneck. The research done by Ng et al. shows that the overhead is even worse for XML-RPC, and the latency gets really bad for complex messages. This could be one of the reasons for the bad result for XML-RPC.

The difference between SOAP and REST are minimal for arrays above 800 objects. For arrays containing below 800 objects, the difference is minimal for all three. When REST is built through Zend Framework, the performance of REST is both good and the latency is stable even for arrays with 3200 objects. The extremely bad result from the first benchmarking of REST implemented in PHP together with the fact that the support library for XML-RPC is not enabled by default makes the use of a framework much more favourable than building a web-service by your own.

To note is that each framework has different implementations for each one of the research items and this research only shows the performance from Zend Framework. A bad implementation inside Zend Framework could also be one of the reasons behind the bad performance of XML-RPC.

5.2 Technology

Table 9 presents technology differences collected in the previous sections from each one of the research items. The following paragraphs describe each row from the table more in detail.

	SOAP	XML-RPC	REST
Structs and arrays	YES	YES	NO
Named structs and arrays	YES	NO	NO
Customized data set	YES	NO	YES
Strict typing	YES	NO	NO
Multiple request handling	YES	YES	NO
Transport protocol independent	YES	YES	YES

Table 9 - Feature table Technology

Data types

This section presents details from table 9; structs and arrays, named structs and arrays, and customized data set.

According to XML-RPC specification [7], it supports 8 data types by default such as *Integer*, *Boolean*, *String*, *Double*, *java.util.Date*, *byte[], java.util.Map, Object[]* and *java.util.List*. There are more data types supported in XML-RPC if the property `enabledForExtensions` is set.

According to a SOAP data type summary [8] SOAP has *Integers*, *Booleans*, *Double* and *Strings*, those are called `XSD:[TYPE]` in the WSDL document. *Array*, *Hash* and *Objects* are all named `XSD:STRUCT`. For mixed types, which could be your own defined type, it should be named `XSD:ANYTYPE`. The data from the interviews shows that SOAP has support for much more customization for the developer such as structs and arrays, character set and data set whereas REST is not that comparable in this case. For REST, data is sent in *String* to the server and then the server responses in various representations accordingly such as HTML, XML, PDF and more.

Strict typing

SOAP requires strict typing which, according to the interviews, often is unnecessary but good for services when correctness is of importance. For example payment solutions or closed systems. XML-RPC on the other hand does not have strict typing and will not break if a client uses a double instead of a float. According to the interviews this strict typing is often not necessary and most web-applications works better without it.

Multiple request handling

Multiple request does not mean handling request in different threads simultaneously, instead, we mean performing more than one method call or procedure by a single request. XML-RPC is extremely good for this purpose since the data transferred is in an specified XML format, which means there are no limits for the client to add more than one method call in that XML. But all method calls are processed in turns on the server not at the same time. This feature also applies to SOAP since XML-RPC and SOAP are very similar in many ways whereas REST in this case is not competitive.

When using HTTP as transport protocol, which is also the most common way, handling multiple requests with REST is not as convenient as the other two mentioned before. This is also stated in the interviews. Since the representative of requested resource is identified by unique URL usually, it is hard to execute more than one method at one time. There are ways to work around it, for instance, designing specific URL to call many methods before representing the resource. Again, it becomes more a design issue from the beginning for building a web service.

Transport protocol

As mentioned in the literature review SOAP is transport protocol independent and according to Allman (12) XML-RPC is as well so both of them could therefore be used in any environment.

Compared to them, REST does not restrict communication to a particular protocol (11), but it does constrain the interface between components, and hence the scope of interaction and implementation assumptions that might

[7] http://ws.apache.org/xmlrpc/types.html
[8] http://old.apisnetworks.com/soap-data-types.php

otherwise be made between components (1). A common misunderstanding discovered during this research is that people think that REST is limited by only using HTTP as protocol. This misunderstanding also appears in the data from the interviews. This may because web can be considered as REST service (20), and for a web service or application, HTTP is the most common protocol to use. According to Pautasso et al. (11) due to the constrains of the limitations that HTTP protocol have, request methods are usually limited by only using "GET" and "POST". Because not all servers allow users to use methods like "PUT", "DELETE" for security reasons. Therefore, additional effort and time are needed to work around (11).

5.3 Usability

Table 10 compares features of SOAP, XML-RPC and REST. It is generated based on data collected and presented in the previous sections.

	SOAP	XML-RPC	REST
Specification	YES	YES	YES
Tools framework	YES	YES	YES
Extensibility/Maintainability	EASY	EASY	EASY
Security	HIGH	HIGH	N/A
Overhead	HIGH	HIGH	LOW
Short learning curve	NO	YES	YES

Table 10 – Feature table Usability

Specification

Based on data from the interviews, SOAP requires very strict matching on both client and server side, which may cause problems and makes the implementation hard. According to the interviews SOAP has a specification that could cause problems. When writing a SOAP client it can be crucial to use the same version of the SOAP specification as the server is written in. According to the interviews the SOAP specification changes often and that can cause problems if the server was written to follow a previous specification.

XML-RPC is very well documented which means when developing it, people can put focus on build the server side functions and simply assume that the XML got from the client follows its specification. This is also well supported in data from the interviews.

As mentioned in Luiz and Celso's paper (20), REST is an architecture, it does not have any specifications for it in terms of developing details. Instead, it is a concept that needs to be understood. REST's resource is identified by URL and its content can be accessed in different formats such as XML, HTML, JPEG etc. However, it is not easy to find such specifications and this resulted in that people think XML is the only representation of data resource according to the interview.

Tools /Framework

As discovered during the benchmarking, PHP has a library to support XML-RPC but it is not enabled by default. That means if it is not enabled on the server you use and if you don't have control over the server you will not be able to use it. According to the interviews it could be useful to use a framework together with XML-RPC.

For SOAP, the WSDL document has been described in the data from the interviews as complex and hard to understand. To note is that there are some WSDL generators available online to help generate the WSDL document based on the written code.

As REST gains more popularity recently, there are some frameworks available to use such as Zend, Restlets [9] and Simple Web [10].

Extensibility/Maintainability

According to the data from the interviews SOAP lacks in maintainability, it is mainly the WSDL document that makes it time consuming and hard to maintain. This could also affect the extensibility. To extend a SOAP server the WSDL document needs to be updated. It is also important to note which

[9] http://www.restlet.org/
[10] http://www.simpleweb.org/

SOAP specification to use and follow when maintain and extend a SOAP application.

XML-RPC on the other hand is said to be easy to use and learn (12), this is also mentioned in the data from the interviews. Maintaining and extending it should therefore not cause any further problems. REST has also been said to have modifiability and extendibility due to its flexibility. On the contrary REST is just an architecture and has been said to have a complex server and deep understanding of its architecture is of course needed.

Security

For SOAP it is the WSDL that binds the server to the client and creates a secure connection (3) without any extra effort from the developer. XML-RPC was built with no intention to protect the data at all. Even passwords were transmitted in clear-text (21). Nowadays it is possible to bind the client to the server and get a secure connection, but this is not automatically done as in SOAP and it needs some extra functions to be used. According to the interviews SOAP is safer due to its strict typing. If the client uses a different data type from the server a SOAP server will create an error when a XML-RPC server will only crash if the developer haven't prepared for that specific fault (13). For REST no adequate information about security was found.

Overhead

As stated in data, both from the interviews and according to Phan et al. (22), both XML-RPC and SOAP suffer from high overhead. This is due to the XML serialization and deserialization and the underlying transport protocol TCP. This has to be considered especially for resource constraint applications. According to Nayef et al. (2), the performance could be increased by storing and reusing the message template. The research done by Chuik et al. (23) reveals that by reducing the number of comparisons for XML tags by using trie data, performance could also be increased.

REST does not have the same problem with overhead due to its implementation flexibility. The overhead added is done through the implementation for each specific application, and not enforced by the protocol.

Short learning curve

According to our own experience when building the benchmarking, the learning curve for building a REST service is relatively low comparing to the other two. HTTP clients and servers are implemented in most of the major languages already and supported by most of the hardware or operating systems (11), therefore the base of building a service is already done. The only thing that one needs to learn is the concept of REST service. To build a service, frameworks like Zend also does great help for build both REST client and server in PHP and it is well documented. Therefore, it is easy to be adopted and inexpensive to acquire (11).

According to the interviews XML-RPC is also easy to use and learn and there are a lot of materials, forums and tutorials available online. For SOAP on the other hand the learning curve is steep, this is also mentioned in the interviews. The data from the interviews reveals that it is the WSDL which is the complicated part to learn. Whether you build a server or a client you have to understand the WDSL document.

6. Conclusions

This paper examines three of the most common ways for data transfer through web-services, SOAP, XML-RPC and REST. Many standards and protocols are available today and when choosing a way to implement a web-service, a lot of aspects have to be considered. These three were therefore examined not only from a performance perspective but also from an implementation view. This was done through benchmarking and interviews. The data collected from these two were analysed in

conjunction to each other and a model is presented.

In short, SOAP should be used for systems when correctness is important but is more complex both to learn and use.

RPC is easy to use and suitable in most cases but performs very poorly on large amount of data.

REST is good for open services and especially for service oriented systems because of its simplicity of identifying resources by URLs and the light weight messaging.

For future research, more and extensive benchmarking could definitely contribute to this research. Besides, deeper studies could be focused on what are the factors that affect the performance for different standards/architecture in the area.

More standards, frameworks or protocol could also be added to this research, SOAP, XML-RPC, REST and Zend are only fours ways of implementing a web-service available today, other are for example JavaRMI or CORBA.

SOAP, XML-RPC and REST could be used in conjunction to each other. It is possible to use REST architecture together with both SOAP and XML-RPC. It is also possible to build a SOAP/ RPC server. These scenarios could also be interesting for a future research.

7. References

1. **Fielding, Thomas Roy.** *Architectural Styles and the Design of Network-based Software Architectures.* Irvine : University of California, 2000.

2. **Abu-Ghazaleh, Nayef, Lewis, Michael J. och Govindaraju, Madhusudhan.** *Differential Serialization for Optimized SOAP Performance.* State University of New York : Department of Computer Science, 2004.

3. **Fremantle, Paul, Weerawarana, Sanjiva och Khalaf, Rania.** *Examining the emerging field of web services and how it is integrated into existing enterprise infrastructures.* New York : ACM, 2002.

4. **Parashar, Manish och Davis, Dan.** *Latency Performance of SOAP Implementations.* The state university of New Jersey : Center for Advanced Information Processing, 2002.

5. **Dissanaike, S., Wijkman, P. och Wijkman, M.** *Utilizing XML-RPC or SOAP on an Embedded system.* Stockholm : Stockholm University, 2004.

6. **Overeinder, B. J, Verkaik, P. D. och Brazier, F.M. T.** *Web Service Access Management for Integration with Agent Systems.* New York, USA : ACM, 2008.

7. **Ng, Alex, Chen, Shiping och Greenfield, Paul.** *An Evaluation of Contemporary Commercial SOAP Implementations.* North Ryde, Australia : u.n.

8. **Toyotaro, Suzumura, o.a.** *Performance Comparison of web service engines in PHP, Java and C.* Tokyo : Tokya Research Laboratory, IBM, 2008.

9. *Verivue - RPC and REST.* **Vinoski, Steve.** u.o. : IEEE Computer Society, 2008.

10. **Fielding, Thomas Roy.** *Principled Design of the Modern Web Architecture.* Irvine : University of California, 2005.

11. **Cesare Pautasso,Olaf Zimmermann,Frank Leymann.** RESTful Web Services vs. “Big” Web Services:Making the Right Architectural Decision. 2008.

12. **Allman, Mark.** *An evaluation of XML-RPC.* New York, USA : ACM, 2003.

13. **Andrew S. Tanenbaum, Robbert van Renesse†.** *A Critique of the Remote Procedure Call Paradigm.* Amsterdam, The Netherlands : Vrije Universiteit, 1988.

14. *IONA Technologies - RPC Under Fire.* **Vinoski, Steve.** u.o. : IEEE Computer Society, 2005.

15. **Cecchet, Emanuel, o.a.** *Performance Comparison of Middleware Architectures for Generating Dynamic Web Content.* New York : Springer-Verlag, 2003.

16. **Amza, Christiana och Emmanuel Cecchet, et. al.** *Specifications and Impelementation of Dynamic Web Site Benchmark.* u.o. : WWC: IEEE 5th Annual workshop on Workload Characterization, 2002.

17. **Head, Michael R., o.a.** *Benchmarking XML Processors for Applications in Gridd Web Services.* New York : ACM, 2006.

18. **Berg, Bruce L.** *Qualitative Research Methods for the Social Science.* Long Beach : California State University, 2007.

19. **Allan, George.** *A critigue of using grounded theory as a research method.* UK : Portsmouth University, 2003.

20. **Luiz Alexandre Hiane da Silva Maciel, Celso Massaki Hirata.** An Optimistic Technique for Transactions Control using REST Architectural Style.

21. **Nelson, Andres D. Birrell and Bruce Jay.** *Implementing Remote Procedure Calls.* Palo Alto, CA : Xerox Palo Alto Research Center, 1984.

22. **Phan, Khoi Anh, Tari, Zahir och Bertok, Peter.** *A Benchmark on SOAP's Transport Protocols Performance For Mobile Applications.* Dijon, France : ACM, 2006.

23. **Chiu, Kenneth, Govindaraju, Madhusudhan och Bramley, Randall.** *Investigating the limits of SOAP performance for Scientific Computing.* Edinburgh, Scotland : Bloomington, Indiana University, 2002.

8. Appendixes

A. Interview Questions

SOAP
For someone that never used SOAP, is it easy to use and learn?
What kind of pre knowledge is needed?
Have you used any official documentation of SOAP?
-Is it easy to find?
-Is it easy to understand?
What are the advantages for SOAP in your opinion?
What are the disadvantages for SOAP in your opinion?
Which are the special occasions when SOAP is better than the other standards to use?
What is your overall impression of SOAP?

XML-RPC
For someone that never used RPC, is it easy to use and learn?
What kind of pre knowledge is needed?
Have you used any official documentation of RPC?
-Is it easy to find?
-Is it easy to understand?
What are the advantages for RPC in your opinion?
What are the disadvantages for RPC in your opinion?
Which are the special occasions when RPC is better than the other standards to use?
What is your overall impression of RPC?

REST
For someone that never used REST, is it easy to use and learn?
What kind of pre knowledge is needed?
Have you used any official documentation of REST?
-Is it easy to find?
-Is it easy to understand?
What are the advantages for REST in your opinion?
What are the disadvantages for REST in your opinion?

Which are the special occasions when REST is better than the other standards to use?
What is your overall impression of REST?

B. Interview Data

SOAP	
id	*interview text*
100	understand the WSDL specification
101	not easy for anyone that never used it
102	it is complex with SOAP
103	had to learn WSDL
104	not easy, could have been easier
105	most people think strict typing is an advantage
106	if you need strict typing it's good
107	in most cases you don't need strict typing
108	It is very complex
109	the SOAP specification keeps changing
110	It is strict typed
111	if you change a data type it might not work
112	was bound to use SOAP
113	the client break if there is something wrong
114	It is annoyingly complex
115	Time consuming to implement
116	SOAP is not good if you deal with a lot of data
117	changing internal things in SOAP is a big thing
118	A WSDL document is commonly used as a specification over the soap request and answer.
119	it makes the SOAP implementation clean and easy
120	to define the WSDL specification some knowledge and research is needed
121	the SOAP implementation itself is not hard but it is the WSDL part that takes time to understand
122	I did not use any official documentation, there are A lot of resources available online. Forums, tutorials
123	its a nightmare
124	Lots of programming language have built in clients and servers but these actually make it harder to use unless you are using the same language on both client and server
125	You need to know services, XML, data types, HTTP
126	There is a spec, its not easy to understand because SOAP itself is complicated
127	If you're using the same language on both client and server it can be very robust and work well
128	It is very complicated
129	data types between languages are often badly handled
130	It isn't an exact spec and the inconsistencies make it very hard for different services to inter operate
131	SOAP is fine, its businessy and lots of clients have built-in clients and servers.
132	Its a buzz word and most implementations are poor.
133	SOAP is slowest because although you can get a system up really quickly, these systems have huge maintenance basic overheads

XML-RPC	
id	*interview text*
200	it depends, easier than SOAP
201	many XML-RPC in general available on the net
202	need knowledge about XML
203	need to basically understand the idea of RPC
204	used a framework that generate the RPC calls
205	it's simple without any complicated types
206	it has a simple structure with strings and ints
207	it should be used for loose web applications
208	not good with closed applications
209	not good if the client should break
210	Choose between XML-RPC and REST
211	XML-RPC is good for open web-services
212	did not use official documentation
213	Very easy to use and learn
214	Pre knowledge about programming
215	I haven't used any official documentation
216	Its easy to understand
217	makes a nice wrapper for an existing set of functions
218	Function and interface are very closely coupled
219	It's good for internal systems where the consumer of the service is already familiar with the functions
220	it's good if you are wrapping existing functions which are documented.
221	Not so useful if designing a service from scratch
222	RPC is Basic but functional
223	RPC is fastest of these three

REST	
id	*interview text*
300	The easiest way
301	how to know how to make an URL
302	it works with all browsers
303	It is only a URL
304	anyone knows how to create an URL
305	it can only use HTTP as transport protocol
306	if I would build an easy service
307	not complex
308	easy to document
309	easy to understand
310	fast to implement
311	if you have many requests REST is not that good
312	REST is very easy
313	REST could be a good idea for the "spel-server"
314	REST is good for open web-services
315	the size of data should be considered when choosing standard
316	I have never built my own, but used REST
317	it is very application specific
318	it's not so complex and if you don't need that it is a good choice
319	often you don't need the complexity in web-apps
320	it's very easy to use, especially for small apps
321	I did a web-interface, for adding, grabbing all kinds of data
322	Yes I think it is easy to learn, compared to the other
323	not that much work but got some help from others
324	some knowledge about the scripting lang you use
325	did not use official documentation
326	used wikipedia and google
327	it's standardized so it's easy for others to use
328	it could be a bit complicated
329	have to spend some extra time for a REST server
330	I think it is good, makes sence
331	Not an standard but an architecture, quite abstract
332	focus on structure rather implementation details
333	it can be built in different ways
334	no built-in support in PHP
335	build both client and server from scratch
336	not that complex because there are examples that you can reach on the internet
337	it is simply to modify and extend due to its flexibility
338	Yes, it is easy to use and learn
339	Pre knowledge is only HTTP
340	There are some specs and I have the O'Reilly RESTful web services book.
341	All documentation is excellent
342	Its clean, simple and uses a lot of build in functionality of HTTP
343	People misunderstand it and write horrible
	interfaces with everything routing through one controller - thereby losing half the advantages
344	It's good when there are other requirements such as security
345	It's good when you want to consume with various languages/clients
346	Love it
347	For systems where performance is important I always use REST