2013-5

A performance analysis of WS-* (SOAP) and RESTful Web Services for Implementing Service and Resource Orientated Architectures

Philip Markey
Technological University Dublin

Gary Clynch
Technological University Dublin

A Comparative Analysis of WS-* (SOAP) & RESTful Web Services for use in Service/Resource Orientated Architecture

Philip Markey, Gary Clynch

Department of Computing, Institute of Technology Tallaght, Tallaght, Dublin 24
phil.markey@gmail.com
gary.clynch@ittdublin.ie

Abstract

Over the past number of years, the architectural styles that have been utilised for distributed computing with the use of Web Services have been developing and evolving, leading to numerous debates as to the suitability of the differing styles. This has led to two main supporting stands being developed, WS- Standards and Representational State Transfer. With this there have been points made against the presumed complexities of the WS-* and favour made towards Representational State Transfer with the usage of the standardised Web API.*

In this dissertation, the author gives an overview of both of the Web Service styles and their associated architectures, as well as presenting experimental results in which to demonstrate the performance of each style, over a number of test scenarios.

Keywords: SOAP, REST, RESTful, Resource Oriented Architecture, Service Oriented Architecture.

1 Introduction

Within the area of distributed computing, two key areas of Service Oriented Architecture (SOA) and Resource Orientated Architecture (ROA) have been viewed as competitive styles, each encompassing differing orientations [10]. The first, SOA, is generally associated with SOAP alongside the WS-* stack and the latter ROA, tends to be associated with the Representational State Transfer (REST) style.

Both approaches are adaptable to accomplish a distributed system but there are points that can be drawn both for and against with each. For example it can be viewed that the possible complexities of the WS-* stack could be viewed negatively when looking towards a comparative of the simplistic use of the native Web Application Programming Interface (API) as used in a RESTful approach. With the same view point, the opposite could also be surmised as, the simplistic use of the Web API, may not be sophisticated enough to be used with an enterprise system and therefore the mature and well documented WS-* stack should be utilised in its place.

This paper has three sections preceding a conclusion. First a brief overview of SOA, ROA, SOAP and REST is given. Then a descriptive is provided of a benchmark environment and testing scenarios, which is finally followed by the result analysis based on the benchmark testing that was conducted.

2 An Exploration into SOA/ROA & Web Services

SOA is a flexible set of design principles, when utilised efficiently can provide a loosely coupled set of services via a single endpoint, that can form one, all-encompassing application that simplifies

machine-to-machine communication [6]. For this SOA services utilise a set of standards that allow applications to be published, discovered and invoked such as the Web Services Description Language (WSDL) [3].

SOA services focus is on providing a schema and message-based interaction with an application [3]. XML is the common message format that is used for SOA services; the SOAP standard is normally adopted containing an XML payload [6].

At a fundamental level, the SOAP-based architecture revolves around the transmission of XML-encoded messages over a transport medium and is utilised in the formal Web Service approach. The specifics of a SOAP service is that it follows a very well defined set of rules that are prescribed for in the WSDL files, which are essentially XML structured files adhering to a W3C-specified grammar [5].

ROAs concepts are based on three points of Resources, URI and Representations, which are provided over multiple endpoints, each linking to a single resource [7]. Resources are objects that can be referenced and stored on a computer; this could be the returning result from a SQL query or a bit stream representation of a document. Representations are simply the state/format of data of the required resource.

The Resource is referenced by the use of a URI [1]. The URI is essentially just the name and address of the requested resource, with best practices should be as descriptive as possible to best match the resource that is being addressed.

REST is a set of specific guidelines that can be covered to produce an implementation of a RESTful service [7]. It can be used to design a Web Service that focuses on system resources, including how resource states are addressed and transferred over HTTP [8].

REST is based on a set of transfer operations that are universal to any data storage and retrieval system. These operations are commonly referred to by the acronym CRUD as shown in *Table 1: Core HTTP CRUD Methods*, which stands for Create, Read, Update, and Delete.

CRUD	**REST**	
CREATE	POST/PUT	Initialise the state of a resource at a given URI
READ	GET	Retrieve the current state of the resource
UPDATE	PUT	Modify the state of the resource
DELETE	DELETE	Clear a resource.

Table 1: Core HTTP CRUD Methods

3 Benchmark – Experiment

3.1 The Benchmark Environment

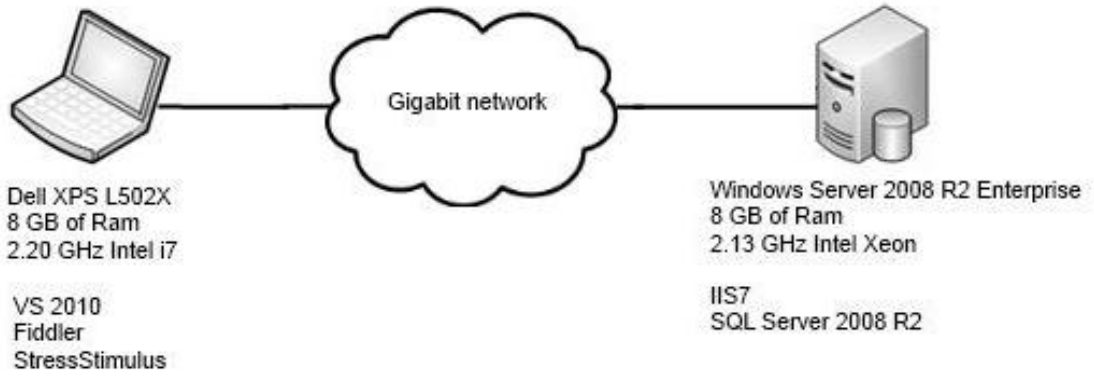

Figure 1: Benchmark Network Topology

As displayed in *Figure 1: Benchmark Network Topology*, the benchmark environment is a simple set up, that consists of a Gigabit network, a server with a Windows Server 2008 R2 Enterprise installation and a Dell XPS.

Within this network set up, the server acts as the Web Service host, hosted in Internet Information Services 7 (IIS7). The server is also running an instance of SQL Server 2008 R2, which is accessible to the Web Services.

On the XPS there is an intermediary piece of software installed that acts as a proxy host, called Fiddler2 [4]. With Fiddler2, HTTP traffic passes through it for traffic in both directions and has the capability to view this traffic as it passes through and provides the ability to be able to inspect the traffic.

To further extend the capabilities of Fiddler2, a plugin called StresStimulus [9] is installed; StresStimulus is a load testing. Using StresStimulus it is possible to record a number of traffic scenarios and then replay the same scenarios under various load patterns, with a varying number of virtual users while monitoring how the site is performing under that load.

In the benchmarking scenarios used in this session, Fiddler2 is used to capture and analyse the traffic and then StresStimulus is used to replay the same request traffic over again for each repeated test. In all test scenarios only one virtual user is ever used.

3.2 Benchmark Scenarios

For the benchmarking session there are two scenarios that are carried out to allow a comparison to be drawn. Firstly, the service calls are used to request an object from each of the respective Web Services. The object is a country record generated from the database on the test server. This scenario is used to demonstrate the size and weight of each request/response and demonstrate the affect the wrapping protocol can have.

The Second scenario comprises of two parts, the first takes a similar format as the first scenario, each service call will be requesting an object from its respective service and the service will be fulfilling

this by use of a call the database. For this scenario the request is for a collection of 249 country objects. In this test the calls are repeated over a set timeframe of one minute, with each call being placed consecutively. This would demonstrate not just the weight of the call and its associated wrapping protocol but the amount of traffic that can be produced by each service with an iterative call.

In the second part, is only applicable to the RESTful service calls with their use of the GET Header in the Web API. In this section the test is run again with the caching abilities enabled. From this, it will again show the weight of the traffic that is transmitted and show the number of calls that could be successfully placed within the given timeframe. Further to this, show what affect enabling caching can have when looking at services request/responses.

4 Results & Analysis

4.1 Test One

In the first test, both of services had a method invoked that interacted with a database via the Entity Framework and retrieved a single item. This item was then packaged up and transported back to the client in the various formats. This was used to demonstrate the differences in size of the sending and returning single payloads.

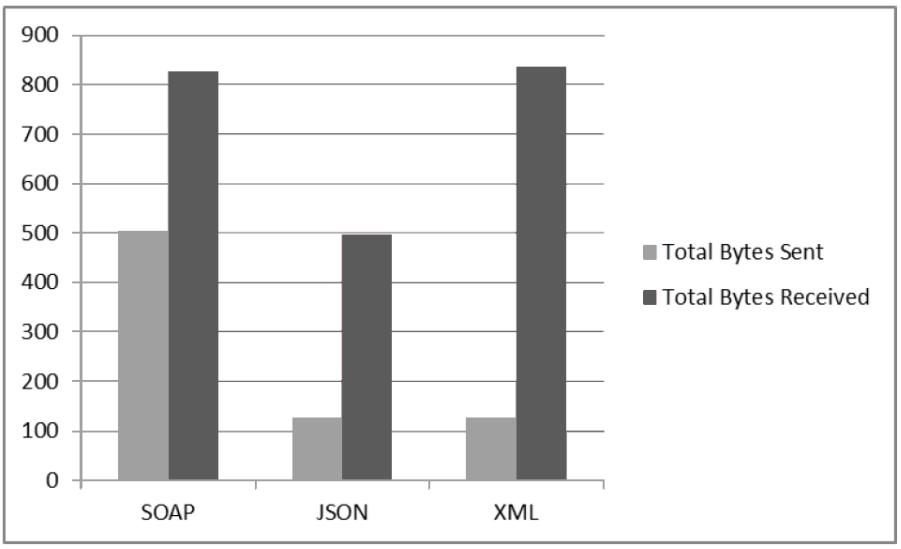

Figure 2: SOAP vs. REST Single DB Item Comparison

	SOAP	JSON	XML
Total Bytes Sent	505	128	127
Total Bytes Received	826	496	836
Total Overall	1331	624	963

Table 2: SOAP vs. REST Single DB Item Comparison

Within this test, shown in *Figure 2: SOAP vs. REST Single DB Item Comparison* and *Table 2: SOAP vs. REST Single DB Item Comparison*, differences can be seen as to the comparative between the SOAP and RESTful calls. SOAP sending 505 bytes, whereas both the JSON and the XML RESTful calls only sending 128 bytes and 127 bytes respectively, this representing only 25% the size of the

SOAP request. On the received data, the XML RESTful call returns an extra 10 bytes compared to the SOAP response but the JSON call received little over half this with only 496 bytes.

In the overall data transported for this test case, the data sent/received by the RESTful XML service is 72% that of the size of the SOAP service but the JSON service is smaller again, with a total data transported representing just 47% that of the SOAP service and 65% that of the XML service.

4.2 **Test Two**

Test two comprised of two parts, firstly both types of service were used to call a method that would return a collection of items from the database (249 items per call) repeatedly over a minute duration, with each call being placed consecutively. After this the test was carried out a second time for the RESTful service, this time with caching enabled.

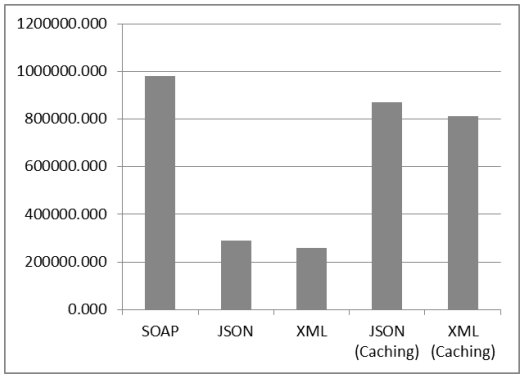

Figure 3: SOAP vs. REST DB Collection Comparison - Bytes Sent/Sec

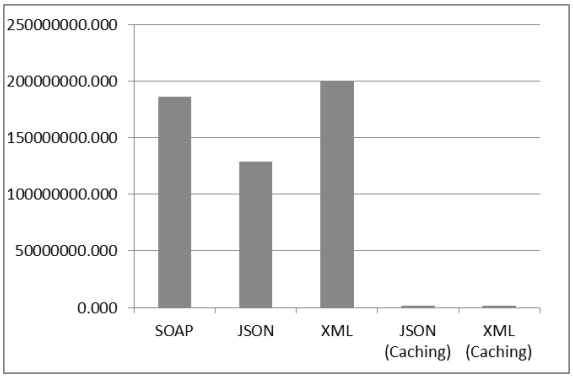

Figure 4: SOAP vs. REST DB Collection Comparison - Bytes Received/Sec

	SOAP	JSON	XML	JSON (Caching)	XML (Caching)
Total Bytes Sent	981682.5	289230	260022	868937	810619
Total Bytes Received	185677951.25	128586060	199181080	961354	1054510
Total Overall	186659633.75	128875290	199441102	1830291	1865129

Table 3: SOAP vs. REST DB *Collection Comparison*

From *Figure 3: SOAP vs. REST DB Collection Comparison - Bytes Sent/Sec*, *Figure 4: SOAP vs. REST DB Collection Comparison - Bytes Received/Sec* and *Table 3: SOAP vs. REST DB Collection Comparison* we can see that the weight of the calls from the SOAP service is relatively heavy in comparison to the RESTful calls and the responses from the XML RESTful service is yet again marginally heavier than it SOAP counterpart as previously noted.

When we take a view of the total bytes sent/received for each call service, it can be noted that in the scenario where caching is not enabled, the RESTful XML call is in fact larger when compared to the SOAP service call by 6.8%, with the JSON service still the smallest at 64.6% the total size of the XML call and this representing 69% that of the SOAP service.

The interesting difference comes with caching is enabled. Although the RESTful requests increase, they are still significantly less than that which is required by the SOAP service. With this, the responses are reduced by a distinct margin due to the server only having to return a header message, which states that no change has occur to the data and a suppressed body. This in turn changes the total

data transported to show that with caching enabled the RESTFul calls are 0.9% that of their SOAP counterpart.

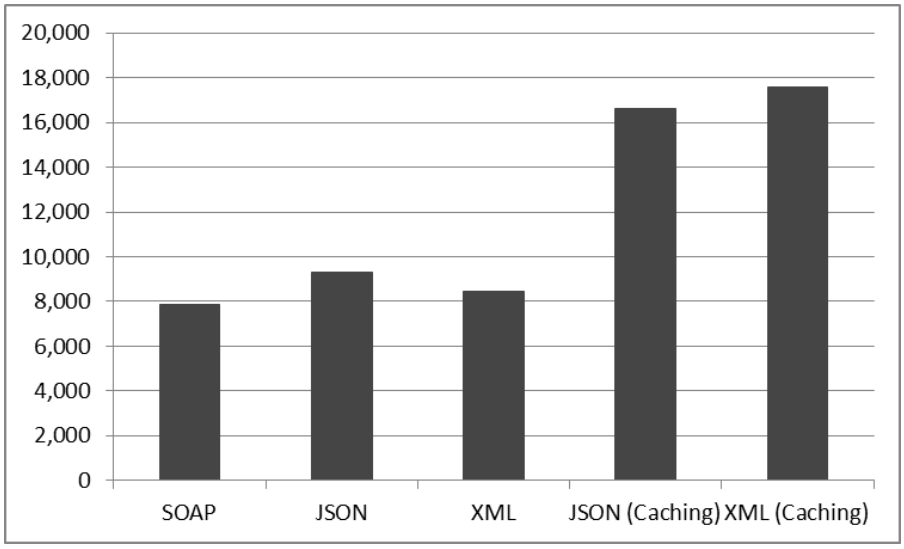

Figure 5: SOAP vs. REST Total Requests Send/Received

	SOAP	JSON	XML	JSON (Caching)	XML (Caching)
Total requests	7,885	9,330	8,456	16,632	17,559

Table 4: SOAP vs. REST Total Requests Send/Received

Also, if we are to look at the total amount of requests that are being sent and received during the minute test, as shown in *Figure 5: SOAP vs. REST Total Requests Send/Received* and *Table 4: SOAP vs. REST Total Requests Send/Received*, both the services were able to process a similar number of calls without caching enable. When the caching was enabled for the RESTful service calls, the service is able to process approximately double the amount from the previous iteration.

4.3 **Analysis**

With both SOAP and REST termed under the same heading of Web Services, they can both stand separately with their architectural styles for their uses in SOA and ROA respectively and each can have significantly differing effects on a network load.

A major point for utilising a RESTful service is the fact that a RESTful service follows the Web API and uses the appropriate headers of GET, POST, PUT and DELETE and therefore it can make full use of a Conditional GET header, enable caching of resources that can improve scalability of a system.

As highlighted the results from the benchmark testing. A RESTful service and in particular when using the messaging format of JSON, can be significantly lighter that its SOAP counterpart when traversing a network.

When the RESTful service comparison is shifted towards the XML messaging format, the results show that without caching, the choice of messaging format could be crucial as the XML format was out performed by both other services, with it being 6.8% larger than the SOAP service and 54.8% larger than its JSON counterpart in relation to total data transported for the scenario.

Once the caching was enabled for the testing, the results show that there would be very little to choose

between the two RESTful services, given that no change occurred to the underlying data. In the scenario, once caching was enabled, this then showed the total weight of data to be transported dropped to approximately 1% that of the SOAP service; which is unable to take advantage of the HTTP caching capabilities.

5 Conclusion

5.1 Conclusion

From the information put forward in this dissertation, as to which would be the more discerning choice, it would appear that due to the low impact of a RESTful service on the transport medium and with the utilisation of the HTTP headers enabling the use of Conditional GET caching, a RESTful service could be selected as the optimal choice and in particular the use of the JSON messaging format. Of course if fully standardised protocol usage was required and the transport impact was not a concern then the only way would be to utilise the WS-* standards.

5.2 Future Work

To further enhance this analysis with future work, we could extend the comparison to also include the development to the RESTful services, by means of also including OData alongside the JSON and XML RESTful services.

Also within an area of drawing a comparison of the WS-* stack and the RESTful services further work could be looked towards a discussion to give complete low level comparison of the both the conceptual and technological concepts that can be seen between the two styles, to be able to give a full quantitative technical comparison basing on the architectural principles that can been drawn between them.

From this lower level view, it then may be possible to present a decision tree that could be used when making a selection on the most suitable usage, based on the initial development requirements. Taking the technical development points and comparing them with the solutions and possible alternatives that may be offered from each of the services. From here it would be possible to add a weighting to finally draw a decision.

6 References

[1] Berners-Lee, T. . (2005) *Uniform Resource Identifier (URI): Generic Syntax* [Online]. Available from: http://labs.apache.org/webarch/uri/rfc/rfc3986.html [accessed: 17/12/11]

[2] Fielding, R.T. (2000) *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.

[3] Meier, J.D., Homer, A., Hill, D., Taylor,J., Bansode, P., Wall, L, Boucher, R. & Bogawat, A. (2008) *Application Architecture Guide 2.0, Patterns & Practices*

[4] Microsoft Corporation. (2009) *Fiddler2* [Online]. Available from: http://www.fiddler2.com/fiddler2/ [accessed: 01/10/12]

[5] Mulligan, G. & Graˇcanin, D. (2009) *A Comparison of SOAP and Rest Implementations of a Service Based interaction Independence Middleware Framework*

[6] Ort, E. (2005) *Service-Oriented Architecture and Web Services: Concepts, Technologies, and*

Tools

[7] Richardson, L. & Ruby, S. (2007) RESTful Web Services

[8] Rodriguez, A. (2008) *RESTful Web Services: The basics*. [Online]. Available from: http://www.ibm.com/developerworks/webservices/library/ws-RESTful/index.html [accessed: 09/10/10]

[9] Stimulus Technology (2011) *StresStimulus* [Online]. Available from: http://stresstimulus.stimulustechnology.com/

[10] Tsai, W. T., Zhou, X. (2008) *SOA Simulation and Verification by Event-driven Policy Enforcement*. pp.165-172, 41st Annual Simulation Symposium 2008