REST as an Alternative to WSRF: A Comparison Based on the WS-Agreement Standard

Florian Feigenbutz, Alexander Stanik, and Andreas Kliem

Technische Universität Berlin, Complex and Distributed IT Systems,
Secr. EN 59, Einsteinufer 17, 10587 Berlin, Germany
{florian.feigenbutz}@campus.tu-berlin.de,
{alexander.stanik,andreas.kliem}@tu-berlin.de
http://www.cit.tu-berlin.de

Abstract. WS-Agreement and WS-Agreement Negotiation are specifications that define a protocol and a language to dynamically negotiate, renegotiate, create and monitor bi-lateral service level agreements in distributed systems. While both specifications are based on the Web Services Resource Framework standard, that allows using stateful SOAP services, the WSAG4J reference implementation additionally provides a RESTful service implementation of the same operations. This paper evaluates the performance disparity between the standard conformable and the RESTful implementation of WS-Agreement and WS-Agreement Negotiation.

Keywords: SLA, WS-Agreement (Negotiation), WSAG4J, REST, WSRF.

1 Introduction

Nowadays, software architects and developers have the fundamental choice between two major approaches when creating web services: WS-* based or RESTful web services [13]. Both acronyms describe popular approaches for distributed services: the WS-* family describes a large stack of specifications based on the *Simple Object Access Protocol (SOAP)* [12] while *Representational State Transfer (REST)* is more an "architectural style" [10] than a standard that strongly relies on the *Hypertext Transfer Protocol (HTTP)* as the application-level protocol [19]. There is a notable number of well defined WS-* specifications which are modularly designed in a way that they can be changed, combined, and used independently of each other [8]. Many of these specifications specify interfaces which are usually defined by the *Web Service Description Language (WSDL)* [6]. With the upcoming of *Web Application Description Language (WADL)* [14] as equivalent to WSDL, such a specification chain can also applied to REST.

This paper investigates to what extent a specific WS-* specification could be ported to RESTful services and be extended with WADLs. Furthermore it studies both the SOAP and the REST based implementation in terms of the feature

B. Benatallah et al. (Eds.): WISE 2014, Part II, LNCS 8787, pp. 294–303, 2014.

set and the performance. For the comparison we use the *WS-Agreement (WSAG)* [4] and the *WS-Agreement Negotiation (WSAN)* [23] specifications which are built on top of the *Web Services Resource Framework (WSRF)* [1] standard. The intention of WSRF is to provide a stateful WS-* web service which can be used to model, access and manage states in distributed systems [11]. Our comparison is based on an existing open source software framework, named *WS-Agreement for Java (WSAG4J)* [22] [24], that implements the WS-Agreement and the WS-Agreement Negotiation standards. The reason for choosing these specifications and this framework is that a significant effort to design a RESTful approach of both specifications was already investigated by [15] [20] [5]. Any WSAG service acts as a neutral component between a service provider and a service consumer for SLA agreement and contracting. As such it needs to be available to both at any time which implies hard requirements for availability and scalability of such a service. Therefore we expect our performance benchmarks to indicate whether WSRF or REST based WSAG allows to handle more concurrent clients with given hardware. Furthermore, the WSAG4J framework itself provides also both a SOAP and a RESTful service [24] with an appropriate client implementation.

The rest of the paper is structured as follows: In section 2 we present our practical approach for this comparison, where performance benchmarks had been performed and show the differences in terms of scalability, availability, and efficiency. For this evaluation a test scenario was designed that respects not only atomic operations but also workflows for which both WS-Agreement standards were conceived. Moreover, we interpret results, discuss the reasons and analyze their origin. Next we present related work, that compares WS-* based services to RESTful ones in section 3. Finally, section 4 concludes this paper.

2 Evaluation

In order to compare both the RESTful and the WSRF variants it is important to select a real life usage scenario with significant complexity. A typical use case for WSAG is automated SLA negotiation which is already used by research projects in the area of fully automated service-level agreements [22] [7]. Web services handling SLAs via WSAG and WSAN act as neutral notaries which must always be reachable to both agreement parties guaranteeing verification of concluded contracts. For this reason performance, scalability and availability are hard requirements for any production system.

2.1 Test Scenarios

Based on the use case of automated SLA agreement we picked three sample tests that reflect WSAG usage in the field of cloud computing. Terminology is adapted from the WS-Agreement and WS-Agreement Negotiation specifications [4] [23]. Basically every WSAG and WSAN service provides at least one *Agreement Factory* containing one or more *Agreement Templates* that describe the

provided services and serve as a sample for incoming *Agreement Offers* the service is willing to accept. *Templates* also hold information for agreement creation such as context or terms and can optionally define creation constraints allowing customization of *Agreement Offers*. *Offers* are created by the *Agreement Initiator* and sent to the *Agreement Responder*. If the latter accepts an *Agreement* is created, otherwise the *responder* replies with a *Fault*. For the given scenarios both WSRF and REST services were configured with one *Agreement Factory* holding three templates and are specified as follow:

- **GetFactories.** The first scenario is a very basic step in which an *Agreement Initiator* requests all *Agreement Factories* served by the *Agreement Responder*. This use case is usually the first step an *initiator* has to go through to discover services of an unknown *responder*.
- **GetTemplates.** The next scenario reflects the follow-up step in service discovery: The *initiator* needs to gain knowledge about available *Agreement Templates* for any *factory* of interest.
- **Negotiation Scenario.** The third scenario runs a complex negotiation process between the *initiator* and the *responder*. In this case the *responder* implements the agreement on behalf of the *Service Provider* while the *initiator* acts on behalf of the *Service Consumer*. The offered service computes resources for certain time frames using negotiable templates. Within the scenario the *initiator* sends a first *Negotiation Counter Offer* to which the *responder* replies with another counter offer for less resources at the same time or an equal amount of resources at a later time. The *initiator* evaluates given options and sends a third counter offer which is finally accepted leading to a *Negotiated Offer* used by the *initiator* to create the offer.

2.2 Test Infrastructure

The load tests used two commodity servers providing four virtual machines as shown in figure 1. Each server was equipped with two Intel Xeon E5430 2.66 GHz CPUs (four cores per CPU) and 32 GB RAM. The nodes were connected via regular Gigabit Ethernet links and ran Linux (kernel version 3.2.0-57). Both nodes ran KVM virtual machines with two cores. Inside the virtual machines we used Ubuntu Linux 12.04 (kernel version 3.2.0-57) and Java 1.6.0.26 (OpenJDK). Tests were coordinated using the Java based load testing framework *The Grinder* in version 3.11 [3].

Host 1 provided *vm1* which ran Apache Tomcat 7.0.50 and served the WSAG4J web apps with a maximum of 2 GB heap space. To allow dedicated usage of available heap space only one of both apps (WSRF and REST) was deployed simultaneously. Host 2 provided *vm2*, *vm3* and *vm4* which executed the test runner component of the Grinder framework named *grinder-agent*. The three agents were coordinated by another host running the Grinder's *grinder-console* component which handled code distribution, test synchronization and collection of measurement results.

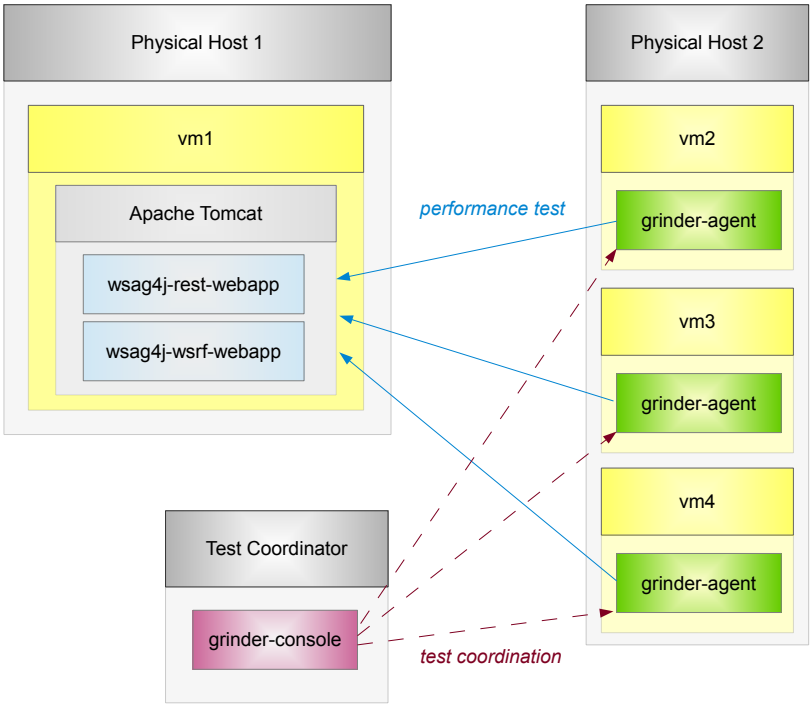

Fig. 1. Physical architecture of test environment

This infrastructure allowed short network paths avoiding biased results due to network issues while still being close enough to real life scenarios in which clients will always be located on different machines than the WSAG service.

2.3 Impact of Security Technology

In the context of SLA negotiation security features such as non-repudiation form the technological foundation for general feasibility and acceptance. We aimed to ensure a comparable level of trust for both approaches: WSRF and REST.

WSAG4J's WSRF based solution utilized WS-Security standards such as *BinarySecurityToken* and *XML signature* [18] [9] by default while the RESTful distribution shipped without adequate replacement. Therefore we chose to run all tests with HTTPS and replaced WSRF's security tokens with *TLS Client Certificates* which verify the identity of request senders. Because message payload was neither encrypted with WSRF nor REST by default we enabled TLS for both variants considering the sensitive nature of SLA agreement to protect communication from any kind of eavesdropping. Nevertheless this setup could not

ensure message integrity if any intermediate host would be able to tamper with the message's content. Given that intercepting a TLS connection would require substantial effort this discrepancy was assessed as negligible for test results.

2.4 Measurement Results

Our load tests measured 200 test runs for each test scenario with both WSRF and REST code bases. Each test scenario was executed with a different number of concurrent clients to evaluate the scalability of both solutions. All tests started with a single client and increased up to 8 concurrent clients. All JVMs of the Grinder agents as well as Apache Tomcat were restarted after each run to minimize effects of JVM internal optimizations.

Figure 2, 3 and 4 show response times of all test scenarios. The results of our load tests reveal that the RESTful code base provides better performance than WSRF in most cases. More specifically there is only two results which show better response times of WSRF: *GetFactories* with 1 and 2 concurrent users. Starting with 4 concurrent users the RESTful stack performs better.

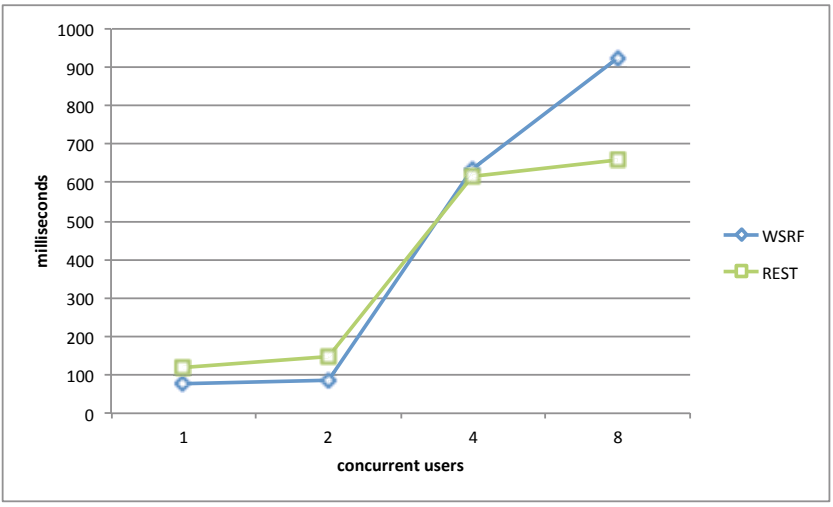

Fig. 2. Response times for GetFactories

For *GetTemplates* and the *Negotiation Scenario* results reveal lower response time of the RESTful approach in all cases. It is important to point out that while running the *Negotiation Scenario* an increasing number of test failures appeared with rising numbers of concurrent users. Using the WSRF stack the first failures appeared with 4 concurrent users and concerned already 80% of all tests while the RESTful stack showed 59% of failures under the same load. This is also the reason why figure 4 only reveals times up to 4 concurrent users. With more than 4

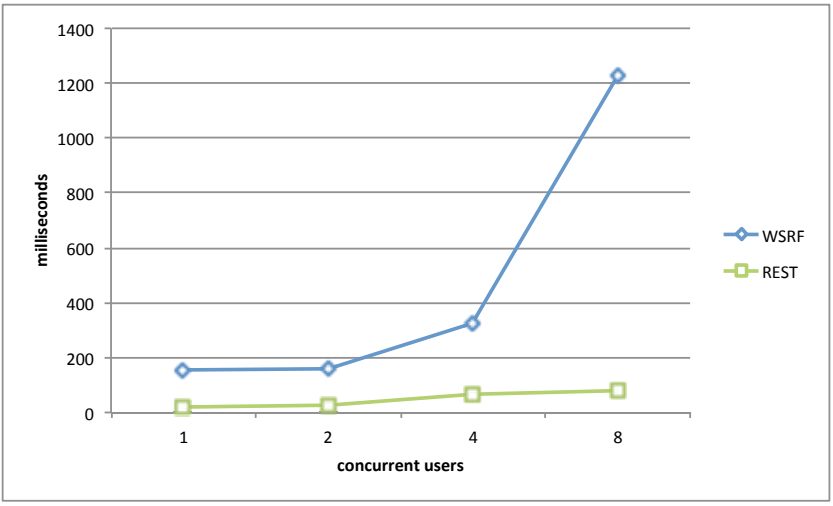

Fig. 3. Response times for GetTemplates

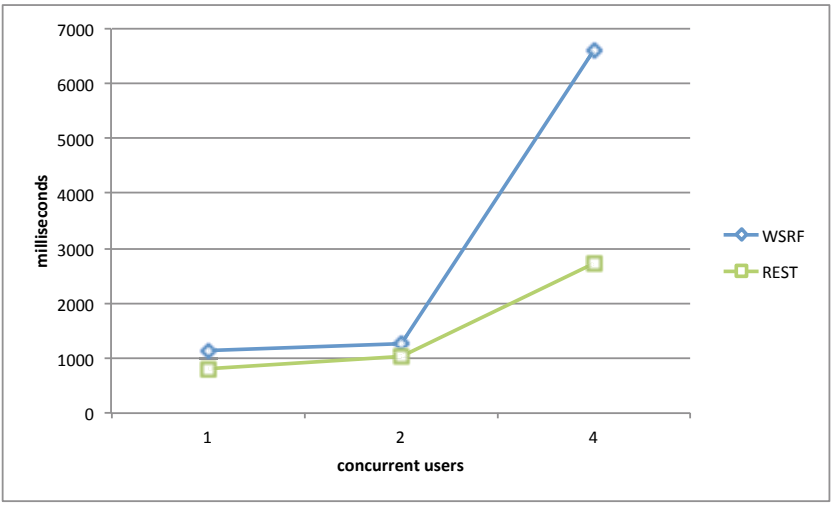

Fig. 4. Response times for NegotiationScenario

concurrent users the number of failures increased rapidly leading to unreliable measurement results.

Last figure 5 compares response times of all test scenarios proving increased complexity of the last scenario in terms of computation time.

Due to the modularity of WSAG4J, the implementation of functionalities for processing agreement offers, creating agreements, monitoring the service quality, and evaluating agreement guarantees is comprised in the SLA *Engine Module*

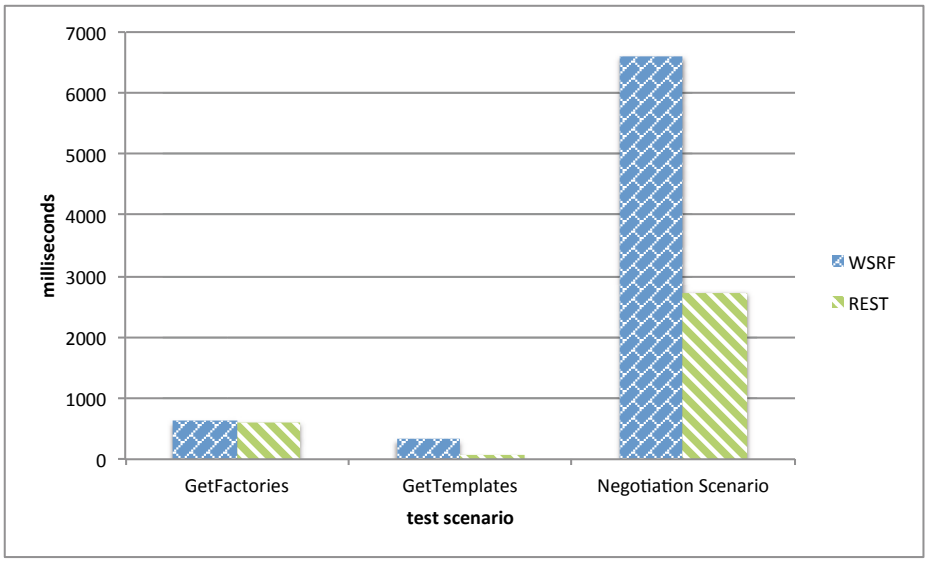

Fig. 5. Response times of all test scenarios with 4 concurrent users

which is used by both web service stacks the WSRF as well as the REST. Therefore we follow the black box approach where we did not separate between performance of internal components like the *Engine Module* and the frontend *Web Service Modules*. We focus on the performance comparison of WSRF to REST where the overhead for parsing the WS-Agreement language, for persistence of agreements or for business logic is the same.

Besides measuring response times we also evaluated the amount of network traffic each solution required during the tests. HTTP request and response in the case of *GetFactories* are compared and show that in this sample case REST required nearly one-tenth of WSRF's network traffic by using the very basic media type *text/uri-list* instead of a more complex and verbose XML structure. Both numbers of 3637 and 378 bytes were aggregated over the relevant payload. In order to compare only payload required for the specific use case, security data such as WS-Security headers or client certificates was neglected.

3 Related Work

The comparison of WS-* respectively SOAP and RESTful web services has already been performed by several scientists [13] [19] [16] [21] [17]. However, the comparison of stateful approaches with the intention to include WADL in a WS-* standard is still an open issue. Thus, the following papers either compared both in different contexts or migrated applications between WS-* and RESTful approaches.

Pautasso et al. [19] compared both WS-* and RESTful web services from a conceptual and technological perspective and developed advices when to use which approach. They presented a general and comprehensive summary to support architectural decisions. In contrast, the focus of this paper is on a specific standard (WSAG) that requires stateful web services by presenting a performance comparison of both approaches.

Upadhyaya et al. [21] provided a semi-automatic approach to migrate existing SOAP based services into RESTful services and compared performance measurements of both solutions showing slightly better performance of REST based services. Compared to their work, our paper focuses on one single WS-* standard and compares already existing services rather than generating them which allows a more detailed inspection of both solutions.

Mulligan and Gračanin [17] developed a middleware component for data transmission offering both a SOAP and a REST interface. They evaluated their implementations with regard to performance and scalability requirements. Other than their work, this paper compares both approaches using very specific WS-* standards: WSRF and WSAG. We also evaluate the performance of both approaches with real life use cases from SLA contracting.

Kübert et al. [15] analyzed the WSRF based WSAG specification and designed a RESTful service with a feature set close to the standard. Their work proved that porting a WSAG service to REST is possible in theory but their scope ended with the proof of feasibility. We use an existing software framework and gain insights about performance gaps between both solutions. Based on their work as well as the existing RESTful implementation of WSAG4J this paper adds an evaluation of both approaches which has not been shown before.

4 Conclusion

In terms of performance it becomes apparent with an increasing number of concurrent clients that the RESTful stack of WSAG4J scales better than the WSRF based solution. In the given test infrastructure we could test the *GetFactories* case with 24 concurrent REST clients without running into failures while WSRF reported 50% failures with a number of 8 concurrent clients. These results are likely to be influenced by the amount of required network traffic which is significantly larger in the case of WSRF and therefore puts a higher load on the latter's serialization engine.

All tests were executed with HTTPS terminated by Apache Tomcat. To enhance the latter's TLS performance future work could include the *Apache Portable Runtime (APR)* [2] to enhance Apache Tomcat's TLS performance. We expect that enabling APR will reduce RESTful response times as the REST setup relies on Tomcat to verify and handle client certificates while WSRF uses its inbuilt logic to handle WS-Security tokens and would therefore profit less from enabling APR.

Finally we underline that web services providing WS-Agreement and WS-Agreement Negotiation act as neutral notaries which must by definition always be reachable to both agreement parties enabling 24/7 verification of SLAs.

As proved by our measurements the RESTful implementation of the WSAG4J framework scales better than the WSRF based solution and can therefore reduce operation costs and complexity when using WSAG4J for SLA negotiation and monitoring.

The second reason for the REST based solution is its enhanced interoperability compared to the WSRF variant. When providing a public WSAG service it is advisable to support as many different clients as possible. Because REST's technological footprint is lighter than the one of WSRF, it is open to more development environments possibly attracting a larger number of users.

As future work we see a higher investigation into specifying RESTful operations for WS-* specifications, especially for the WS-Agreement and the WS-Agreement Negotiation standards. This is also one of the hot discussed topics of the Grid Resource Allocation and Agreement Protocol Working Group (GRAAP-WG) of the Open Grid Forum (OGF).

Acknowledgment. The research leading to these information and results was partially supported by received funding from the European Commission's Competitveness and Innovation Programme (CIP-ICT-PSP.2012.5.2) under the grant agreement number 325192. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the MO-BIZZ project or the European Commission.

References

1. Web services resource 1.2 (ws-resource) (April 2006), http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf
2. The apache software foundation: Apache portal runtime (2013), http://apr.apache.org
3. The grinder, a java load testing framework (2013), http://grinder.sourceforge.net
4. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web services agreement specification (ws-agreement) (March 2007), http://www.ogf.org/documents/GFD.192.pdf (updated version 2011)
5. Blumel, F., Metsch, T., Papaspyrou, A.: A restful approach to service level agreements for cloud environments. In: 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC), pp. 650–657 (December 2011)
6. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., et al.: Web services description language (wsdl) 1.1 (2001), http://www.w3.org/TR/wsdl
7. Comuzzi, M., Spanoudakis, G.: Dynamic set-up of monitoring infrastructures for service based systems. In: Proceedings of the 2010 ACM Symposium on Applied Computing, SAC 2010, pp. 2414–2421. ACM, New York (2010), http://doi.acm.org/10.1145/1774088.1774591
8. Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Unraveling the web services web: an introduction to soap, wsdl, and uddi. IEEE Internet Computing 6(2), 86–93 (2002)

9. Eastlake, D., Reagle, J.: Xml signature (2000), http://www.w3.org/Signature/
10. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California (2000), AAI9980887
11. Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D., Tuecke, S.: Modeling and managing state in distributed systems: The role of ogsi and wsrf. Proceedings of the IEEE 93(3), 604–612 (2005)
12. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J., Nielsen, H.F., Karmarkar, A., Lafon, Y.: Simple object access protocol (soap) 1.2 (2002), http://www.w3.org/TR/soap/
13. Guinard, D., Ion, I., Mayer, S.: In search of an internet of things service architecture: Rest or ws-*? a developers perspective. In: Puiatti, A., Gu, T. (eds.) MobiQuitous 2011. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 104, pp. 326–337. Springer, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-30973-1_32
14. Hadley, M.J.: Web application description language (wadl) specification (2009), http://www.w3.org/Submission/wadl/
15. Kübert, R., Katsaros, G., Wang, T.: A restful implementation of the ws-agreement specification. In: Proceedings of the Second International Workshop on RESTful Design, WS-REST 2011, pp. 67–72. ACM, New York (2011), http://doi.acm.org/10.1145/1967428.1967444
16. Muehlen, M., Nickerson, J.V., Swenson, K.D.: Developing web services choreography standards the case of {REST} vs. {SOAP}. Decision Support Systems 40(1), 9–29 (2005), http://www.sciencedirect.com/science/article/pii/S0167923604000612 WS-REST 2011
17. Mulligan, G., Gracanin, D.: A comparison of soap and rest implementations of a service based interaction independence middleware framework. In: Proceedings of the 2009 Winter Simulation Conference (WSC), pp. 1423–1432 (2009)
18. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R.: Web services security: Soap message security 1.0 (2004), http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
19. Pautasso, C., Zimmermann, O., Leymann, F.: Restful web services vs. "big"' web services: Making the right architectural decision. In: Proceedings of the 17th International Conference on World Wide Web, WWW 2008, pp. 805–814. ACM, New York (2008), http://doi.acm.org/10.1145/1367497.1367606
20. Stamou, K., Aubert, J., Gateau, B., Morin, J.H.: Preliminary requirements on trusted third parties for service transactions in cloud environments. In: 2013 46th Hawaii International Conference on System Sciences (HICSS), pp. 4976–4983 (January 2013)
21. Upadhyaya, B., Zou, Y., Xiao, H., Ng, J., Lau, A.: Migration of soap-based services to restful services. In: 2011 13th IEEE International Symposium on Web Systems Evolution (WSE), pp. 105–114 (2011)
22. Wäldrich, O.: Orchestration of Resources in Distributed, Heterogeneous Grid Environments Using Dynamic Service Level Agreements. Ph.D. thesis, Technische Universität Dortmund, Sankt Augustin (December 2011)
23. Wäldrich, O., Battre, D., Brazier, F., Clark, K., Oey, M., Papaspyrou, A., Wieder, P., Ziegler, W.: Ws-agreement negotiation version 1.0 (March 2011), http://www.ogf.org/documents/GFD.193.pdf
24. Wäldrich, O., et al.: Wsag4j: Web service agreement for java, http://wsag4j.sourceforge.net, version 2.0, Project Website