Published in: Alt, R., Fähnrich, K.-P., Franczyk, B. (Eds.), Proceedings First International Symposium on Services Science ISSS'09, Logos, Berlin, 2009 (www.logos-verlag.de)

REST and Resource-Oriented Architecture

Wilson A. Higashino[1], M. Beatriz Felgar de Toledo[2], Miriam A. M. Capretz[3]

[1]Venturus - Innovation & Technology Center, Av. José de Souza Campos, 900 - 10th floor13092-123, Campinas/SP, Brazil
[2]University of Campinas, Av. Albert Einstein, 1261 - Cidade Universitária13083-970, Campinas/SP, Brazil
[3]University of Western Ontario London, Ontario, N6A 5B9, Canada

wilson.higashino@venturus.org.br, beatriz@ic.unicamp.br, mcapretz@eng.uwo.ca

Abstract: REST is an architectural style created for hypermedia distributed systems, such as the Web. Recently, REST has been successfully used to create Web Services. Supporters of this style argue that the paradigm is simple as compared to the sheer quantity of specifications and the complexity of the traditional Web Services that are based on SOAP. This article reviews the REST style and the REST-based architecture, Resource Oriented Architecture, to build Web Services. Moreover, it also compares different aspects of the RESTful and WS-* based Web Services. Finally, we review the current situation of this new paradigm, presenting applications, discussion themes and shortcomings.

1 Introduction

Web Services that are based on SOAP technology are well-consolidated and have been enabling the development of applications in many areas, such as business process composition and execution. In spite of its relative success, SOAP-based services have been heavily criticized because of their complexity and their large number of related specifications. Recently, a new alternative to Web Services implementation has emerged. The so-called RESTful Web Services propose the development of services based on the same principles that guide the 'Human-Web' and ensure its success.

This article aims at describing RESTful Web Services and presenting the strengths and weaknesses of this new paradigm, especially in terms of how it contrasts with traditional Web Services. Section 2 reviews the concepts of the REST architectural style. Section 3 explains how the REST style could be applied to the construction of services. Next, section 4 compares the services development paradigms through different perspectives, and in section 5, applications based on RESTful Web Services and the situation of this paradigm are presented. Finally, section 6 concludes the article and presents directions for future research.

2 Basic Concepts

During the first half of the 1990s, the explosion of the World Wide Web (WWW) was evident. In this early stage of development, the Web architecture was only described by a set of informal documents, many of which were outdated in relation to their implementations. The REST architectural style presented in [Fielding 2000] was part of the efforts for standardization of the Web architecture and protocols. Accordingly, the main objective of this work was to create an architectural style for hypermedia distributed systems in order to establish the architectural principles that should govern the Web, evaluate the existing implementations of Web elements and create an evaluation framework for new proposals and functionalities. Furthermore, REST was developed in order to account for low entry barriers, extensibility, distribution, anarchic scalability and independent deployment. A set of constraints was sequentially applied to the Web architectural elements to derive this style: client-server, stateless interactions, cache, uniform interface, layered system and on-demand code. Of these six constraints, the uniform interface has had the greatest impact. The concepts used to define interfaces in the REST style include the resource, the representation and hypermedia as the engine of application.

- A *resource* is any information that can be named, such as, a document, an image or a list of open issues in a software version. A resource is formally described as "a temporally varying membership function $M_r(t)$ which for time t maps to a set of entities, or values, which are equivalent. The values in the set may be resources representations and/or resources identifiers" [Fielding 2000].
- A *representation* is defined as "a sequence of bytes, plus representation metadata to describe those bytes" [Fielding 2000]. Specifically, the client receives a representation when it requests a resource or sends one when it wishes to update a resource.
- The last concept is *hypermedia as the engine of application*. This means that the client is responsible for maintaining the application state and for transiting between possible states. The representation of the resources should have hyperlinks indicating states close to the current one, helping applications to perform state transitions.

3 Resource Oriented Architecture

Applying the REST style to Web Services development involves using the same principles that make the human web successful at implementing automatic services. The term 'RESTful Web Services' is generally used to designate Web Services based on this style. In [Richardson/Ruby 2007], the authors define a new architecture, known as Resource Oriented Architecture (ROA), to develop RESTful Web Services. In the ROA, REST principles are explicitly associated with

Web protocols, such as URIs, HTTP and XML. Following, the example of a photo service is used in order to illustrate the ROA. Basically, there are two entities on this service: photos and albums.

Obviously, the main concept of ROA is *resources*. In this architecture, a resource has a name and an address represented by a URI (Universal Resource Identifier). URIs provide a way to uniquely identify a resource, regardless of its type and representation. If the photo service was hosted at the address www.example.com, the photo album '2008Holiday' would have the following URI: http://www.example.com/2008Holiday. An application is considered addressable if it exposes the interesting aspects of its data set as resources [Richardson/Ruby 2007]. The extensive use of URIs provides addressability for the ROA Web Services. This characteristic, along with statelessness, implies that the possible states of applications should also have a URI.

As described with reference to the REST style, clients and servers communicate in ROA through resource *representations*. An important aspect of ROA is that different representations of a resource should have different URIs. This structure enhances the visibility of the architecture because all of the required information for choosing representations is already on the URI.

Hypermedia as the Engine of Application also is an important feature in ROA. In our example, the photo album representation would include the URIs of the photos that belong to it. In ROA, this property is called connectedness [Richardson/Ruby 2007]. ROA explicitly associates the REST constraint of uniform interface with HTTP and its methods and tries to fully respect their semantic. Thereby, to obtain resource data, an HTTP request with the GET method needs to be sent to the resource. A resource deletion could be done through an HTTP DELETE request. The other HTTP methods that could be used include HEAD, OPTIONS, PUT and POST. Response codes are another feature of HTTP protocol used in ROA that defines the operations semantics. The HTTP defines a set of response codes for client requests, such as 200, meaning 'Ok', and 404, indicating that the resource was not found. These same codes can also be used to indicate operation results and services exceptions.

4 ROA vs. WS-*

The traditional way to implement Web Services is based on the protocols of SOAP [Gudgin et al. 2007], WSDL [Christensen et al. 2001] and the WS-* stack, which includes WS-Addressing [Gudgin et al. 2006], WS-ReliableMessaging [OASIS 2004] and WS-Security [OASIS 2006b]. There are several ways of comparing the WS-* stack and ROA which can be found in the literature. According to [Snell 2004], the main difference between ROA and the WS-* stack is that the former utilizes a resource-oriented approach, while the latter perspective is activity-oriented. In ROA, the fundamental application resources are exposed to clients, which can use a set of uniform operations to manage them. On the other

hand, the WS-* stack exposes activities that handle resources, which clients cannot directly see. Using the photo album example, in the ROA, the application would expose resources such as albums and photos. Conversely, with the WS-* stack approach, there would be a 'copyPhoto' operation, whose parameters would include the photo name and the destination album. The service implementation is responsible for manipulating the resource representations. The author of [Snell 2004] believes that both paradigms can coexist; the decision to choose one over the other basically depends on the nature of the application.

The work presented in [Richardson/Ruby 2007] compares the paradigms listing the protocols and functionalities of the WS-* stack and the RESTful Web Services alternatives to these protocols. The comparison starts with SOAP and the way in which it is used. Normally, SOAP implementations send their requests encapsulated in HTTP POST requests and addressed to endpoints, which represent the services with which they communicate. Information about the executed operation and its scope are found inside the SOAP envelope. This limited use of HTTP makes it appear as merely an envelope and transport media, eliminating some of its possible benefits. However, ROA takes full advantage of HTTP by using its functionalities as part of the application, benefiting from REST style characteristics.

In [Vinoski 2008], the author compares the approaches of specific interface definitions with the REST uniform interface. All APIs exposed through the WS-* stack have their own unique vocabulary, complicating the semantic knowledge for intermediaries and clients. Moreover, the specificity of a contract, as is the case in WS-* stack, can inhibit its reuse. In ROA, the methods and response codes of HTTP compose the service interface, which uniformizes the access and increases the system transparency, reducing complexity and promoting reuse.

The work in [Pautasso et al. 2008] compares the two styles based on the concept of architectural decision models. The authors compare the number of required decisions and the alternatives for each decision in the WS-* stack and REST based developments. Generally, each model entails a comparable number of decisions, but the WS-* stack has a large number of alternatives for each decision. The article stipulates that much of the WS-* stack complexity derives from the great number of existing specifications. The apparent simplicity of REST is explained by the fact that REST "removes the need for making a series of further architectural decisions related to the various layers of the WS-* stack and makes such complexity appear as superfluous". Since these specifications are mostly optional, they can be eliminated in order to compare the paradigms, which make them "two technology-level variants of the same conceptual design".

In [Muehlen et al. 2005], REST and SOAP are compared as design philosophies of Web Services choreography standards. The authors suggest that both approaches can solve the problems, but the choice between them in the standardization process is not only based on technical arguments.

5 Current Situation and Applications

RESTful Web Services are relatively recent and consequently, there is no consensus on the best practices and principles that should be used for their architectures and implementations. The lack of standardization can be considered a weakness of the REST paradigm, as even the scholars and supporters of REST disagree on some aspects. For instance, the proponents of 'Lo-REST_ argue that only the GET and POST HTTP methods should be used, since not all proxies and Web servers fully understand the other methods and there may be firewalls configured to block HTTP requests which use them [Pautasso et al. 2008].

5.1 Representations

Another point of controversy involves the representation format used in the communication between the client and the server. Currently, XML [Bray 2006] is the most widely accepted and used format. However, this format is also known for its high overhead and text processing complexity. Therefore, newer and simpler alternatives such as JSON [Crockford 2006] have been emerging. XHTML is another standard language that is used for resource representation [W3C 2002]. One advantage of this format is that the services responses can be directly rendered in browsers. However, XHTML aims at defining information layout, and consequently, is not always appropriate for general representations. In order to solve this problem, XHTML can be used with microformats [Microformats 2008].

5.2 Contracts

The definition of contracts is another point of debate. Many of today's RESTful services only provide a textual description of the access API. This description is often enough because these services tend to be simpler than the services described by WSDL. However, this limited textual description prevents the creation of tools that generate client source codes and verify the correctness of requests. As a way to describe these services, WSDL Version 2.0 provides functionalities to encapsulate RESTful requests in SOAP operations [Chinnici et al. 2007]. This encapsulation hides the REST resource-oriented primitives in activity-oriented abstractions. The WADL (Web Application Description Language) [Hadley 2006] is a more purist alternative. This language was created specifically to describe HTTP based services.

5.3 QoS

Transactions, security and reliable messaging are other aspects of RESTful Web Services that lack standardization. Generally, security aspects are managed through the mechanisms of HTTP, such as digest-based authentication and information confidentiality with SSL. More advanced aspects, such as federation, are implemented through customized solutions. WS-Security [OASIS 2006b], the security specification of the WS-* stack, supports federation as well as several other functionalities. Customized solutions are also necessary for transactions and reliable messaging in the REST context. The WS-* stack supports transactions through WS-AtomicTransaction [OASIS 2007] and WS-BusinessActivity [OASIS 2006a] specifications. Similarly, reliable messaging is managed in WS-ReliableMessaging [OASIS 2004].

5.4 Applications

In the context of REST applications, the Atom specification [Nottingham/Sayre 2005] and its companion AtomPub [Gregorio/Hora 2007] have been gaining increasing prominence. Atom is an RSS-based format which was defined by IETF to read and write information on the Web. This specification defines an XML language aimed at distributing content that can be used with news, blog entries and calendars. Likewise, the Atom Publishing Protocol specification defines a set of HTTPbased services that manipulate Atom. Frequently, this specification is used as a model to build RESTful Web services. Moreover, specific applications extend the Atom format and use the publishing operations to define their service interfaces. Their main advantage lies in their ability to reuse Atom and AtomPub implementations for their own benefit.

Despite all these barriers, there is a substantial number of Web Services based on REST principles, some of which are illustrated in Table 1. However, not all of these services respect all ROA principles. Nevertheless, many of these services are classified as REST as they are based on HTTP. More details about these applications and their perfomance are not provided due to space limitations. A more complete list of RESTful services can be found in [Programmable Web 2008]. An important common aspect of these applications is the simplicity of their clients. In general, clients of RESTful services are much simpler than the WS-* ones.

5.5 Service Composition and Discovery

The efficient nature of REST is highly compatible with the so-called Mashups, a recent concept related to Web 2.0, which consists of creating Web applications by using services from different sources. The work in [Curbera et al. 2007] serves as a platform focused on building these Mashups completely on the basis of RESTful

services. In [Pautasso 2008], the authors propose BPEL language extensions that permit the orchestration of RESTful Web Services. The study in [Pautasso/Alonso 2004] introduces the JOpera, which is a service composition tool dealing with many kinds of activities, including RESTful Web Services.

Not only is service composition critical for Web Services, but dynamic service discovery is also very important. In theory, this functionality allows the runtime selection of a service provider based on pre-defined criteria. In the WS-* stack, the UDDI protocol is responsible for this selection. RESTful Web Services do not have a standard approach to service discovery because their resources integrate the Web and, as such, they can be found through the same approaches used to find information on the Web.

Provider	Service	Type	Description
[Amazon 2008]	Associates Web Service	REST/SOAP	Marketing service
	Simple Storage Service	REST/SOAP	File storage system
	SimpleDB	REST/SOAP	Online database
[Google 2008]	Spreadsheet	REST (Atom)	View and edit spreadsheets
	Code Search	REST (Atom)	Search for function definition and sample code in public sources
	Static Maps API	REST	Map image retrieval
	OpenSocial	REST	Standard services for social networks interaction
[Microsoft 2008]	Windows Live Contacts	REST	Windows Live contacts access
	Windows Live Photo API	REST (Atom/ WebDAV)	View and update Windows Live photos and albums
[Yahoo 2008]	Flickr	REST/SOAP	Organize and share photos
	Web Search	REST	Traditional, contextual, suggestions and ortography searches

Table 1: Examples of REST Based Applications

5.6 Discussion

RESTful Web Services have been sufficiently flexible to implement services of many different natures, as shown in Table 1. These examples are services provided by large corporations, which demonstrates the growth and importance of the paradigm. According to [O'Reilly 2003], the REST interface corresponds to 85% of the usage of Amazon Web Services. As previously stated, the growth is mainly due to the simplicity of creating clients for service consumption. Almost all programming platforms can send and receive HTTP requests. The flexibility of this

format even allows Web Browsers to test services. In the WS-* stack, the use of tools to generate SOAP-based clients helps to simplify the development process. However, these tools are still inadequate, as they are complex and require the knowledge and configuration of many functionalities.

All of the services presented in Table 1 are popular on the Web, but they do not provide a QoS guarantee to their clients. This lack of guarantee is also apparent for the Web Services; therefore, important requirements in enterprise environments, such as transactions and bounded response time, are not supported by these services. Nevertheless, the goal is to provide exactly the same services with which users are familiar for autonomic consumption. As such, companies encourage the use of their services in an increasing number of applications. The use of HTTP is also aligned with new application development trends, such as Ajax (Asynchronous Javascript and XML). In Ajax, client-side Javascript code makes asynchronous requests to URIs in order to obtain data for presentation. The main objective of this approach is to avoid the request-response cycle presented in Web applications. In this context, the same services used by Ajax applications can be consumed by clients of any other nature.

The reuse of existing HTTP infrastructure is another important advantage of RESTful Web Services. The ability to cache requests and perform stateless interactions is essential to the scalability, availability and robustness of large Web sites, and similarly, these same benefits directly apply to RESTful Services. The discussion of best practices and the lack of standardization on interface definition and QoS are the greatest challenges of the REST paradigm. These problems can prevent the use of RESTful Web Services in complex integration scenarios where interoperability and QoS requirements are important. Moreover, some applications are naturally expressed by activities and operations, which can inhibit the resource-oriented approach. Table 2 summarizes the advantages and disadvantages of RESTful Web Services.

Advantages	**Disadvantages**	
Simplicity / Agility	Lack of references	
Flexibility	Less tools support	
Ease of clients implementation	Lack of standardization	HTTP methods representations contracts QoS
Ease of reuse		
Ease of request caches		
Scalability		
Uniform interface		

Table 2: Pros and Cons of RESTful Web Services

6 Conclusions

REST is an architectural style created to describe a hypermedia distributed system, such as the Web. Recently, REST has also been applied to Web Services development and the term 'REST' has started to be used for characterizing any Service based on Web protocols. The Resource-Oriented Architecture is an example of a service architecture based on the REST style. The protocols used in these services are well-known and their deployed infrastructures are almost pervasive. Moreover, they are based on very simple principles, which induce a set of important properties, such as scalability, transparency and platform independence. As a result, the development of RESTful Web Services has been an excellent alternative to the complexity of the WS-* stack.

This article has presented the concepts and characteristics of RESTful Web Services and examined its current situation. As we have shown, there are many areas where there is no standardization or consensus regarding the approach to be applied. Important aspects of Quality of Service, transactions, security and reliable messaging, are not supported by REST. Other important characteristics, such as service composition and contract definition, are still being studied and require further development. The challenge for RESTful Services is to eliminate these problems without losing their inherent simplicity. Regarding the comparison betwen RESTful and WS-* Web Services, the literature still lacks more detailed information about usage statistics and empirical comparisons. Future research would include comparisons of performance and scalability, as well as measurements of how QoS impacts the services. The study of some existing RESTful applications and a review of available tools are also planned.

The main contributions of this article were to review the important aspects of the REST architectural style and the Resource Oriented Architecture, to compare their approaches with those of the WS-* stack, to discuss the benefits and obstacles related to the style and to present applications that already use REST and ROA.

References

Amazon, Amazon Web Services. http://aws.amazon.com/, 07/2008.

Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., Extensible Markup Language 1.0 (Fourth Edition). W3C, 16.08.2006, http://www.w3.org/TR/2006/REC-xml-20060816, 07/2008.

Chinnici, R., Moreau, J., Ryman, A., Weerawarana, S., Web Services Description Language, (WSDL) Version 2.0 Part 1: Core Language, W3C, 26.06.2007, http://www.w3.org/TR/2007/REC-wsdl20-20070626, 07/2008.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., Web Services Description Language, (WSDL) 1.1, W3C, 15.03.2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 07/2008.

Crockford, D., RFC 4627 - The Application/json Media Type for JavaScript Object Notation (JSON), IETF, 07/2006, http://www.ietf.org/rfc/rfc4627.txt, 07/2008.

Curbera, F., Duftler, M., Khalaf, R., Lovell, D., Bite: Workflow Composition for the Web, in: Krämer, B.J., Lin, K.-J., Narasimhan, P. (Eds.), Service-Oriented Computing (ICSOC 2007), Springer, Berlin 2007; pp. 94-106.

Fielding, R., Architectural Styles and the Design of Network-based Software Architectures, PhD thesis, University of California, Irvine 2000.

Google, Google Code, http://code.google.com/, 07/2008.

Gregorio, J., de Hora., B., RFC 5023 - The Atom Publishing Protocol, IETF, 10/2007, http://www.ietf.org/rfc/rfc5023.txt, 07/2008.

Gudgin, M., Hadley, M., Rogers, T., Web Services Addressing 1.0 - Core. W3C, 05/2006, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509, 07/2008.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H.F., Karmarkar, A., Lafon, Y., SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C, 27.04.2007, http://www.w3.org/TR/2007/REC-soap12-part1-20070427, 07/2008.

Hadley, M.J., Web Application Description Language (WADL), 2006, http://wadl.dev.java.net, 07/2008.

Microformats, About Microformats. http://microformats.org/, 07/2008.

Microsoft, Windows Live SDK, http://msdn.microsoft.com/en-us/library/bb264574.aspx, accessed on 10/2008.

Muehlen, M., Nickerson, J.V., Swenson, K.D., Developing Web Services Choreography Standards: The Case of REST vs. SOAP, in: Decision Support Systems, 40 (2005) 1, pp. 9-29.

Nottingham, M., Sayre, R., RFC 4287 - The Atom Syndication Format, IETF, 12/2005, http://www.ietf.org/rfc/rfc4287.txt, 07/2008.

OASIS, Web Services Atomic Transaction 1.1, 07/2007, http://docs.oasis-open.org/wstx/wsat/2006/06, 07/2008.

OASIS, Web Services Business Activity 1.1, 01/2006, http://docs.oasis-open.org/wstx/wsba/2006/06, 07/2008.

OASIS, Web Services Reliable Messaging 1.1, 11/2004, http://docs.oasis-open.org/wsrm/wsreliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf, 07/2008.

OASIS, Web Services Security: SOAP Message Security 1.1, 01.02.2006, http://www.oasisopen.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 07/2008.

O'Reilly, T., Rest vs. SOAP at Amazon, (2003), http://www.oreillynet.com/pub/wlg/3005, 10/2008.

Pautasso, C., BPEL for REST, in: Proceedings 7th International Conference on Business Process Management, Milan 2008.

Pautasso, C., Alonso, G., JOpera: A Toolkit for Efficient Visual Composition of Web Services, in: International Journal of Electronic Commerce, 9 (2004) 2, pp. 107-141.

Pautasso, C., Zimmerman, O., Leymann, F., RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision, in: Proceedings 17th International World Wide Web Conference, ACM, Beijing 2008; pp. 805-814.

Programmable Web, APIs Dashboard, http://www.programmableweb.com/apis, 06/2008.

Richardson, L., Ruby, S., RESTful Web Services, O'Reilly, Sebastopol 2007.

Snell, J., Resource-Oriented vs. Activity-oriented Web services, in: IBM DeveloperWorks, 12.10.2004, http://www.ibm.com/developerworks/xml/library/ws-restvsoap, 06/2008.

Vinoski, S., Serendipitous Reuse, in: IEEE Internet Computing, 12 (2008) 1, pp. 84-87.

W3C, XHTML 1.0 - The Extensible HyperText Markup Language, 2nd ed., 01.08.2002, http://www.w3.org/TR/2002/REC-xhtml1-20020801, 07/2008.

Yahoo, Yahoo Developer Network, http://developer.yahoo.com/, 10/2008.