REST：An Alternative to RPC for Web Services Architecture

Xinyang Feng
Information Engineering University
Zhengzhou Henan, China
hermit2005@sina.com

Jianjing Shen
Information Engineering University
Zhengzhou Henan, China

Ying Fan
Henan Radio & TV University
Zhengzhou Henan, China

***Abstract*—Currently most Web services architectures adopt RPC as their architectural style. But because of the complexity of RPC, there are bottlenecks of RPC-style Web services in Web-scale applications. REST not only can make full use of Web features, but also has the advantage of simplicity. So REST becomes a new alternative to RPC for Web services architecture. In this paper, at first the brief introductions of RPC and REST are provided. Then two kinds of architectural styles are analyzed and compared from the perspectives of scalability, coupling, and security. In the end the development trend of Web services architecture is prospected.**

Keywords- RPC; REST; Web Services; Architectural Styles

I. Introduction

As a new distributed computing technology, Web Services are more and more often used for interoperation among heterogeneous platform and enterprise application integration. Currently, a large number of Web services are based on WS-* (SOAP, WSDL, WS-Addressing, WS-ReliableMessaging, WS-Security, etc.), which are RPC-style. These RPC-style Web services are able to meet the need of small or-medium-scale interoperation and data sharing. Along with the unceasing expansion of application scale, some defects of RPC-style Web services in scalability, complexity and performance are revealed gradually. Limited by RPC style, these defects are hard to overcome. In order to solve above problems, REST (REpresentational State Transfer) comes into sight of Web services developers.

II. RPC Overview

RPC roots in the distributed objects movement, which decades ago began replacing in-memory object messaging with cross-network object messaging in object-oriented applications [1]. So RPC looks the server as a set of procedures and the client calls these procedures to fulfill a task. Procedures are verbal. Therefore the modeling based on RPC is verb-centralized, which is actually a kind of transaction script [2].

SOA is a typical RPC-style architecture. The goal of SOA is to avoid isolated applications by separating business rules and policies into distributed services that applications can share. The main ideal behind SOA is valuable: abstracting business services from application, defining service contracts (e.g. WSDL), and registering service into services repository so as to be queried and shared by the client (e.g. UDDI). According to service contracts, the client can call services remotely by using SOAP messages. And thus a system which is easy to create, maintain and expand is generated. Everything seems so perfect, however merely in theory. The services in SOA are abstracted based on RPC. They are made up of several discrete procedures (operations), so unconsciously it limits the way in which service consumers use these services. But there are countless business rules in actual application systems. If you wanted to cover all business rules with such procedures in services, the cost would be enormous. Furthermore, with the increasing of system scales a series of problems regarding scalability and performance will arise**,** which make the system can not keep a well designed structure. Because RPC-style Web services have high complexity and rely heavily on significant platform, some scholars call this kind of services “large Web services” [3].

III. REST Overview

REST is an architectural style that Roy T. Fielding firstly defined in his doctoral thesis. Fielding participated in developing HTTP1.0. He also was the primary architect of HTTP 1.1, and authored the uniform resource identifier (URI) generic syntax. He saw REST as a way to help communicate the basic concepts underlying the Web. To understand REST, it is necessary to understand the definition of resource, representation and state. A resource can be anything. A resource may be a physical object or an abstract concept. As long as something is important enough to be referenced as a thing itself, it can be exposed as a resource. Usually a resource is something that can be stored on a computer and represented as a stream of bits. A representation is any useful information about the state of a resource. A resource may have multiple different representations. In REST there are two types of state. One is resource state which is information about a resource, and the other is application state, which is information about the path the client has taken through the application. Resource state stays on the server and application state only lives on the client. REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems[4]. Let's take a look at some key constraints in REST:

1. Everything being resource;
2. Resource identification through URI;
3. Uniform interface;

978-1-4244-5160-9/09/$26.00 ©2009 IEEE

4. Manipulation of resources through representations;

5. Self-descriptive messages;

6. Stateless interactions;

7. Hypermedia as the engine of application state.

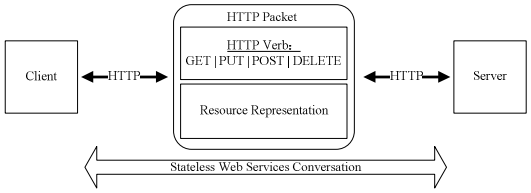

Figure 1 RESTful Web services architecture

As well as abstract principle is concerned, REST is significantly different from RPC. In REST the server is abstracted into a set of resources. A resource is a nominal concept, so the modeling based on REST is noun-centralized, which is a domain model [2]. Figure 1 gives up the illustration of RESTful Web services architecture. The client interacts with the server through uniform interface, and during the stateless interaction the server and the client exchange resource representations.

IV. Comparative Analysis

REST is a coordinated set of architectural constraints that restricts the roles or features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style. Every constraint above mentioned brings some architectural properties for REST, and these properties, such as scalability, low coupling, addressability, are keys to the success of Web. In the following chapters, a comparative analysis on these properties is made between REST and RPC.

A. Scalability

In RPC-style architectures, for example SOA, the service is composed of fine-grained user-defined operations. And different services have different interfaces. Each interface holds its own semantics and operating parameters. So the interface contract is critical to service definitions. If the client wants to interoperate with Web services in SOA correctly, they have to understand the semantics of each interface contract. This approach is no trouble in a relatively closed application environment. But in an open distributed environment, for the unforeseeable number of operations, the approach may cause problems of tightly coupling and interface complexity. These problems prevent the distributed system from having Web-scale scalability. For the current number of Web sites in Web, imagine that if each Web site defines its own special interface and ask the Web browser to download or write a plug-in to adapt to its interfaces. Otherwise, the Web browser will not understand the semantics of the interface and can not interact with the particular site. Then Web browser on the client has to install millions of different plug-ins, which is unacceptable.

REST takes a different way from RPC in the matter of interfaces, and that is uniform interface constraint (constraint3). It means that all resources expose the same interfaces to the client in REST architecture. The most known REST implementation is HTTP protocol. REST is formed based on truly understanding HTTP protocol. In REST, HTTP is a state transfer protocol other than a data transport protocol but. HTTP not only can uniquely locate a resource, but also tell us how to operate the resource. REST demands that a request and response would be performed through four HTTP operations: GET, POST, PUT and DELETE. GET retrieves the current state of a resource in some representation, PUT creates a new resource, POST transfers a new state onto a resource, and DELETE removes an existing resource. Each method has an expected semantics to decide which methods are suitable for each resource. GET is safe, because a GET request will not change the server state. GET, PUT, and DELETE are idempotent, because the three methods being used several times to a resource has the same effect as being used only one time. Uniform interface becomes the Babel of interface definition. It helps to decouple between the client and the server and makes the client and the server can evolve independently. As long as the interface remains unchanged, the client and the serve can interact normally. Uniform interface constraint shifts the variability from interface to resource representation, which makes our concern closer to the ultimate goal -data.

Stateless interactions constrain (constrain6) can also enhance the scalability of RESTful architecture. Stateless interactions constrain demands that each client's request should contain all application states necessary to understand that request. None of state information is kept on the server, and none of it is implied by previous requests. Stateless interactions reduce the cost of enlarging system scale. Because all conversations are stateless, when system scale is up, the only thing needed to do is plugging more load-balanced servers. The coordination between different servers is not needed. Some RPC conversations are stateful. When facing the same question, RPC has to use some additional technologies to achieve the same effect, such as duplicate data or shared memory.

B. Coupling

Service contract of RPC-style architecture defines not only the specialized service interface, but also the data formats involved in interaction. CORBA use IDL to define data formats, SOA generally uses XML Schema to define data formats in WSDL. The manner of binding the interface and data contract is for the convenient code generation. Object-oriented languages such as Java, C++ usually defines data types in the merged way. In these languages, if a data type changes, all callers that use the data type have to be recompiled to make the new definition work. This way goes back on the design principle of splitting interface from implementation and makes the client and the server have tight coupling. Maybe for local applications the negative effect of tight coupling is not obvious, but in Web-scale distributed applications it is the thing we must avoid.

In REST, due to the uniform interface constraint, data formats are orthogonal to interfaces. The resources are manipulated through representations (constraint4) and REST requires messages to be self-descriptive (constraint5). The data formats in representation can change. The representation

formats are based on the result of content negotiation between the client and the server. Different message can also specify different formats in HTTP content-type and accept headers, — the former indicates the message's data-payload format, whereas the latter specifies as part of a request what data formats the caller is prepared to receive in response [5]. The way to handle data format in REST is helpful to reduce coupling between the client and the server. Allowing services to handle multiple data formats means that the client and the service can select appropriate data formats for different types of data, such as images, text, and spreadsheets. Such media types are specific forms of common representation metadata in REST architecture. Using these metadata in messages also makes the client can request their favorite data formats. In HTTP messages, data formats can be identified by MIME. MIME is an international standard that established by IANA [6]. Adopting MIME as the unified data format standard can eliminate ambiguity about data format definitions efficiently.

C. Security

SOAP is widely used in RPC-style Web services now. The client and the server interact by exchanging SOAP packages in RPC-style Web services architecture. SOAP packages are generally wrapped into HTTP envelopes and use HTTP POST to transfer, which make SOAP messages can easily pass the firewall using port 80. This way that just takes HTTP as a tunneling protocol will bring security risks. For the real intention of every request is sealed in the SOAP package, only when the SOAP package is parsed can the server understand its meaning. So if SOAP messages carry some dangerous requests such as illegal delete or maliciously modified, these requests will not be blocked by firewall. To block these requests with potential dangerousness, firewall has to add additional protocol filtering.

Compared with RPC, REST security model is more simple and effective. Everything is abstracted into resource in REST (constraint1), and every resource is identified through its unique URI (constraint2). With these two constraints, if it is necessary to hide some resource, just do not release its URI. REST uses the standard verbs of HTTP, and each verb has clear semantics. Moreover there is no other nesting like SOAP. Setting different permissions for four operations of a resource can make up different security policies. If certain resource is read only, then only GET is exposed to the client; if one is allowed to be removed, DELETE is exposed to the client. These settings can all be completed on HTTP firewall. The difficulty of implementing security policy is greatly reduced.

D. Addressability

Addressability is an important feature of Web application. With addressability users can describe concisely and precisely the information that they want to access. RPC does not reject addressability, and SOA also support URI, however, its naming method lacks a consistent standard. How to name and identify services entirely is up to the designer or developer of the system. And its identifier refers not a resource but a service contract containing a group of operations. The identifier is more like a kind of distributed network handle. Most RPC-style Web services expose very few identifiers, even as few as one. REST requires that URIs should be defined for every resource or resource representation. URI encapsulates all information required to address, and has readability, is also permitted to spread via hyperlinks. There are innumerable URIs in RESTful architecture, which is in consistent with the naming method of Web sites.

E. Connectedness

Connectedness is based on addressability. As a fundamental technology of Web, connectedness forms Web into an interconnected whole, and users can navigate among the Web sites they are interested in. REST requires manipulating resources through representations (constraint4) and taking hypermedia as an engine of application state (constraint7). The server can guide the transfer of application states by links and forms given in the representation. During this process links play an important role in connecting resources. If Web services are well connected, service customers can make a path through the application by following the links and filling out forms. Comparatively, RPC architecture is not good at connectedness. The connectedness in Web means the links between relative data information, while RPC architecture is full of procedures. So, even the URIs of service contracts are linked via some technology, which is a sequence of operations. The purpose of information navigation would not be achieved.

F. Performance

Performance is one of the advantages of REST. First of all, the performance advantage comes from its inherent simplicity of REST. REST is based on existing standards widely used in Web and requires no additional standards, which avoids the dependence on special significant platform and decreases occupancy of system resource. For instance, REST directly uses HTTP protocol in data exchange. In this way the drain on performance of parsing and packaging SOAP packages is avoided on both sides of the client and the server, as well as message payload is decreased. Secondly REST recommends that the client or intermediaries should cache responses that servers mark as cacheable to eliminate some unnecessary interaction for better performance. In actual application, Amazon.com provides both Web services based on REST and SOAP. And according to Amazon's words RESTful Web services run six times faster than ones based SOAP [7].

V. Conclusion

REST is a new mode of thought for service abstraction. It helps to truly understand the original look of HTTP and fully utilize current Web features. REST advocates that Web services should work in a style native to Web. It makes the information available under the same rules that govern well-designed Web sites. So RESTful Web services are "in" the Web instead of just "on" the Web. Compared to RPC-style Web services architecture, RESTful Web services architecture is better in scalability, coupling and performance. With more developing tools supporting REST, it will be more and more adopted and gradually become the mainstream technology of Web services.

References

[1]Steve Vinoski. REST Eye for the SOA Guy. http://www2.computer.org/portal/web/csdl/abs/html/mags/ic/2007/01/w1082.

htm. 2008
[2]Martin Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley. 2002.
[3]Leonard Richardson, Sam Ruby. RESTful Web Services. O'Reilly Media. 2007.
[4]Roy Thomas Fielding. Architectural Styles and the Design of Network-Based Software Architectures. Doctorial Dissertation, Dept. of Computer Science, Univ. of California, Irvine. 2000
[5]Steve Vinoski. Demystifying RESTful Data Coupling. http://www2.computer.org/portal/web/csdl/abs/html/mags/ic/2008/02/mic2008020087.htm. 2008
[6]IANA. MIME Media Types. http://www.iana.org/assignments/media-types/. 2007
[7]Adam Trachtenberg. PHP Web Services without SOAP. http://www.onlamp.com/pub/a/php/2003/10/30/amazon_rest.html. 2003