The Concept of Representational State Transfer (REST)

By Steven Chen

For CEIT, REST API for Remote Laboratories

Introduction to REST

REST is a software architecture that was developed from noting standout features after post hoc analysis of the largely successful World Wide Web and then documented in Roy Fielding's doctoral thesis [1]. It outlines how the components of large distributed hypermedia systems (severs, clients, proxies etc.) should interact with one another, specifically, how resources are requested and passed from servers to clients [1].

Resources and Representations

A resource on the web is essentially anything that is substantial enough to be given its own identification [2]. Clients that request to access a resource are presented with a representation of that resource. Each resource may have one or many representations which are suitable for a variety of client types [3]. For example, a web resource may provide a representation with appropriate layout and styling for web browsers that people are using as well as a machine-readable representation for other web applications to access [3].

Goals of the REST Architecture

Some of the features important to an information network are the scalability of component interactions, generality of interfaces and independent deployment [1]. It is also important that any intermediary components such as firewalls for security and caches to reduce network latency [1] can be operated between servers and client without impacting on availability of information.

In order to ensure that design of a web API is truly RESTful, there are some constraints it must adhere to.

Uniform Interface

The driving force of behind any RESTful service is the uniform interface it must provide. Resources must be identified with universally accepted descriptors otherwise known as Uniform Resource Identifiers (URIs) [2]. When a client requests a resource it must presents the resource's URI which contains the name of the resource as well as the location of it allowing for request of access. Upon receiving a request message, servers will send a representation of resource to the client.

After obtaining that resource representation, the client should have enough information to perform, with permission, modification or deletion of resource elements contained on the server.

The messages received by either client or server must contain information on how to "read" the message and whether the message is cacheable.

Finally, hypermedia is to be the engine of the application state [1]. What this means is that the web application must be designed so that there are hyperlinks available at all application stages which will allow client to request a transition to the next stage. In other words each state is a resource with an URI.

Other REST Constraints

The servers which hold the resources must not be concerned with user interface or state. Meanwhile, clients should not worry about data storage. The result is more scalable servers and clients with improved portability.

The communication client and server must not mention anything about the client state. The fact that all resources will be pointed to by a unique URI means that clients will be able to request for resources pertaining to its next state without the need to let the server know its current state. Benefits of this setup is that website monitoring can be performed [4] to ensure that resource inaccessibility can be detected and dealt with more rapidly as well as improving overall scalability.

Cacheability must be made explicit with each message between server and client so that old or incompatible data is not used in response to further requests [1].

The concept of a layered system [1] is used in REST so that there can be working intermediary components within a network. Ordinarily, in a layered system, the client will not know that communication from the server has passed through other components.

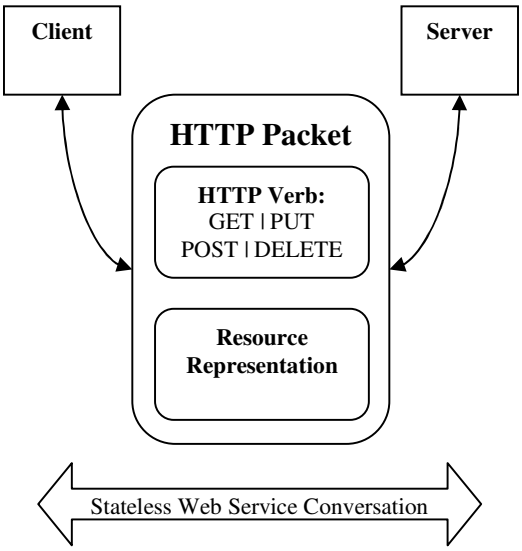

Figure 1 - RESTful Web Services Architecture with HTTP [2]

HTTP is a communication protocol that is often used when creating a REST API although it is not a requirement. The reason being that HTTP provides a rich vocabulary allowing the usage of URIs, "verbs" like GET and PUT, Internet media types, request and response codes, which are quite compatible with REST conditions [4].

Figure 1 shows an example of a RESTful web API which uses HTTP messages. When the client request for a resource (i.e. GET a particular URI) the server will respond with a resource. Communication occurs via a uniform interface and by exchanging resource representations. HTTP verbs are included so server and client will know how to handle the messages they receive.

REST versus SOAP-RPC

A Remote Procedure Call (RPC) is the alternate architecture for design web APIs. Having been developed for the purpose of cross-network

communication of objects belonging to an object-oriented application, RPC provides a web service by allow the client to send procedure call to the server [5] often utilising HTTP as a transport mechanism (POST command [4]). The server is "seen" as a set of procedures which will run according to the message sent [2].

In the case of SOAP-RPC, SOAP messages will be sent between the client and server. Like other types of RPC, SOAP-RPC will "tunnel" new application-specific RPC interfaces through an underlying generic interface [6]. What happens is a channel is opened to transfer system behaviours and application semantics to the client. Since system behaviours are very difficult to prescribe in a distributed environment, applications created with SOAP-RPC are not interoperable by nature [6] which goes against the original intension of web services: wide scale interaction with each other. REST APIs, on the other hand, does not have this problem since the client does not need to understand the system and application semantics.

Every time a client needs to interact with a SOAP-RPC web service, it needs to form a contract in order to begin understanding the interface semantics. In a small scale network, this will not typically pose a problem. In Web-scale distributed systems, however, the extra overhead will be become more and more significant due to a growing number of tight coupling to be made and complex interfaces to understand [2]. In the case of the REST architecture, all interfaces are uniform which means scalability is much greater.

As mentioned previously stateless interactions in a RESTful service means that the server does not need to store any state information i.e. no synchronisation with client or other servers required. This means that it is relatively easy to increase the number of components in a network since there will not be any mass transferring of data to accommodate new additions. The same cannot always be said for RPC because stateful communication is not restricted.

Finally, performance is another area where REST beats RPC. REST is based on existing standards widely used in the Web and requires no additional standards, which avoids the dependence on special significant platforms and decreases occupancy of system resources [6]. Using standard HTTP protocol to exchange data rather than just a transport protocol for exchanging complex procedure calls and system semantics means that REST APIs are faster.

Conclusion

As shown in this report, REST services have a much lower overhead than SOAP-RPC services resulting in better performance. In a large network, it avoids the risk of having to download a large number of plug-ins/add-ons in order to support special platforms associated with RPC. REST is an architecture truly suited for large distributed hypermedia systems.

References

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

[2] X. Feng, J. Shen, and Y. Fan, "REST: An alternative to RPC for Web services architecture," *Future Information Networks, 2009. ICFIN 2009*, pp.7-10, 14-17 Oct. 2009

[3] R. Tomakyo, "How I Explained REST to My Wife", Dec 2004. [Online]. Available: http://tomayko.com/writings/rest-to-my-wife. [Accessed: 7 Aug 2010]

[4] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly, 2007.

[5] P. Mironela, "The Importance of Web Services Using the RPC and REST Architecture," *2009 International Conference on Computer Technology and Development*, pp. 377-379

[6] G. Goth, "Critics Say Web Services Need a REST," IEEE Computer Society, vol. 5, no. 12, Dec 2004