An overview of REST

Jan Newmarch
Centre for ICT
Box Hill Institute

The REST approach was developed to describe the architecture of distributed resource access, such as the WEB. REST in comparison with WS_*stack works with the Web rather than against it and is getting increasing support of not only from developers but also from vendors. This paper explains the philosophy of REST and highlights its simplicity in accessing resources.

Introduction

Since the dawn of networking, computer scientists have been trying to develop frameworks and paradigms for large scale distributed systems. The internet itself provides the backbone for transport and routing, while protocols such as TCP and UDP provide application components with the means of communicating at the application layer. Without a doubt, the WEB has proven itself to be the most successful distributed information system, with other systems such as email running a close second.

Applications such as the Web and email essentially involve human interaction as principal component. But there has been much effort devoted to purely machine-to-machine systems, primarily so that business to business transactions can be conducted across the network. Historically, this has evolved from socket communication, to remote procedure calls, to remote object method calls and to downloadable mobile objects. These are typified by Sun's RPC, CORBA and Java RMI respectively.

Web Services, using technologies such as SOAP, WSDL and UDDI are the current flavour for building machine-to-machine systems. There are many claims that Web Services are a significant advance in such technologies, but equally there are many claims that these particular technologies are in fact a step backwards and that by breaking the conceptual basis of the Web their use will in the long term be damaging to the Web.

Critics need an alternative: such an alternative has existed in conceptual form for many years, and is known as REST (Representional State Transfer). In this article we discuss the basis of REST and how it is usually a more appropriate solution to machine-to-machine transactions than Web Services. Our arguments are based on software engineering, and attempt to avoid any other biases.

Procedure call semantics

In building any kind of system, a program must be written. Programs are complex and in attempting to give them more manageable structure computer scientists have devised procedures, functions and (within the O/O paradigm) method calls. All of these take parameters and return results.

A key to the success of this structuring is that parameters can be divided into *value* and *reference* parameters. With value parameters, the current value of a variable (a simple type, a structure, an array) is passed to a procedure and a local copy is operated on. With reference parameters, the *address* is passed in and changes can be made to this external address within the procedure. There are many variations on reference parameters: Pascal *var* parameters, C passing of explicit addresses, tcl's *upvar* and Ada's *in/out* mechanism (which defers writes until exit from the procedure).

Whatever the mechanism, there is an essential difference between the *value* of a variable and the *address* of the variable. Even O/O languages offer the same distinction: for example, in Java you pass a simple value or the address of an object.

HTML

HTML documents as presented to a browser or other user agents such as a search engine or portal contain a mixture of text, markup and hypertext references. In effect, the text and markup are the *value* of the document and hypertext references are the *addresses* of other documents. This mixture of value and reference is a significant component in what makes the Web successful: a user can read the content supplied and follow embedded addresses to other documents of interest. If everything was by value, then the retrieval of any document would fetch the whole Web – a silly proposition!

XML as a successor to HTML/SGML contains the value component through the text of an XML document. References are also built into many XML document types: for example, Docbook has the tag *ulink* while Xlink defined the attribute *xlink.*

HTTP

The PhD thesis by Fielding (Fielding, 2000) identified further characteristics behind the design of the Web: The Web is built of resources, and each resource has an address (a URI). Fetching that address returns a *representation* of what is at that address. The *value* of an *address* is some piece of data (typically HTML) that represents the resource. The actual resource may be a copy of a static document stored on disk or some data constructed by lengthy computation. That doesn't matter too much: what is important is that the Web deals with two types of concept: a representation equivalent to a value and URI's equivalent to references.

Fielding also identified *verbs* associated with addresses. In languages such as Java and C# it has become a design pattern to distinguish data access methods as *getter* methods to retrieve the value, *setter* methods to set a new value while any other methods have unknown semantics and are a little frowned upon. In the HTTP protocol (W3C, 2008) underlying the Web Fielding identified the *GET* method as a getter method, the *PUT* method as the setter method, while the *POST* method should probably be deprecated as a wildcard method with unknown and arbitrary semantics.

REST does not talk about HTML. Nevertheless, HTML is complementary to HTTP: an HTML document returned as the representation of a URI contains both values and references. This corresponds to Fielding's “layered system constraints” principle, such that a user agent only sees the layer with which it is interacting and the “deeper” layers are hidden within the HTML document.

REST services

Due to increasing security concerns, many web sites are closing inbound accesses to any but HTTP ports (usually port 80 for HTTP, port 443 for HHTPS and proxy ports). Many services have used this to piggyback other protocols into a site even when they are not suitable for HTTP. Examples of services using HTTP or HTTP ports for purposes not necessarily in conformance to HTTP include RMI over HTTP, Skype, and as discussed below, SOAP.

The REST philosophy (Fielding, 2000) is that services using HTTP should conform to the semantics of the Web. That is, they should use representations and URIs and requests should use the HTTP verbs in the manner for which they were designed.

REST is not an API, is not a programming platform and is not a toolkit. It is simply an expression of a design philosophy for distributed applications, and in particular a set of guidelines for designing applications to run over HTTP. Many authors have used these principles in various ways to produce REST or RESTlike architectures. “RESTful Web Services” is an excellent book on a particular architecture embodying REST principles, although one may of course vary components of this architecture according to your own needs.

SOAP

SOAP (http://www.w3.org/TR/soap/) is in flagrant disregard of the REST philosophy and this has consequences: if Web Services based on SOAP actually turn out to be significantly widespread, then they will use significant overheads which a better designed system could avoid. For example, proxy caching simply won't work properly, and “bookmarking” of interesting services can't happen.

SOAP *servers* have an address, a URI. SOAP *services* don't. You can't talk about the address of a SOAP service since it doesn't have one. The effective address of a SOAP service is buried down in the XML that accompanies a SOAP request to a server.

SOAP returns everything by value: it can't return the address of interesting things such as a service, since they don't have addresses. Half of the value of the procedure call abstraction has been discarded by SOAP. Web Service Addressing is a belated attempt to patch an addressing system that didn't need to be broken in the first place.

HTTP has several verbs used in the requests from user agents to servers. The GET calls are meant to be idempotent, causing no explicit action in the server by the request. The server *may* make changes (such as updating a counter), but is not requested to do so. GET calls are thus like getter methods, asking for a value. The responses to GET calls may be cached by proxies along the way, reducing network load for repeated requests. HTTP PUT calls are equivalent to setter methods, and are expected to upload a new value of a resource. POST requests have arbitrary semantics: they may or may not cause changes in the resource. It is impossible for an intermediate agent to know what a POST call will do since the data causing any changes is buried inside the request and only has meaning to the destination server. Responses to PUT and POST calls cannot be cached.

SOAP calls all use HTTP POST. This is the case for getter, setter and other methods. Proxies must always pass the request to the server, even if the request is really just a getter request. In this way, SOAP breaks caching.

Fielding has been noted as saying: “SOAP doesn't just suck, it was designed to suck.” This is essentially because of the issues noted above.

SOAP is also absurdly complex for most uses. In many cases, much simpler request/response formats can be used. For example, SOAP is used by UPnP as well as for Web Services, and an examination of a REST approach to UpnP by the author showed significant space and time savings by discarding SOAP in favour of a more appropriate style.

Using REST

Let's consider a simple example of a REST approach to services. We want to query an organisation for a list of staff and information such as phone numbers. We should look at what we want to label as resources, and what should be the representation of the resource. Information about each staff member is a complex object with many attributes, probably stored in a backend database so it should be a resource. As a resource, it must have at least one URI so that we can access it. In fact each staff member could have a number of addresses according to different attributes within the organisation:

```
/staff/Fred
/postion/manager/Fred
/section/marketing/Fred
```

These different URIs can all refer to the same resource, Fred.

A *representation* of Fred is a piece of information about Fred that is returned to a client requesting one of the URIs for Fred. The representation can depend on the URI alone, or also on the extra information accompanying the request. For example, an HTTP request may specify the (human) language to be used in the response. If the language is specified to be English, then the response data should be in English, but if the language is specified to be Latin, then the response should be in Latin if possible.

What about individual attributes within a person's record? For example, the telephone number? Well, that depends on who is asking. If it is an outsider wanting to know contact information then the value will be enough, but if it is the telephone operator who has authority to update the number then that could be returned as an address. The REST model returns a representation of the resource, and this representation can be customised to the recipient, based on the URI plus other information in the request.

So for example, an external query for a staff list might be

```
GET /stafflist
```

and could return an XML document

```
<stafflist>
  <staff>
    <url link="http://XYZ//staff/Fred"> Fred </url>
  </staff>
  ...
</stafflist>
```

while

```
GET /staff/Fred
```

could return a basic XML document about Fred

```
<person>
  <name> Fred </name>
  <phone> 123456 </phone>
</person>
```

On the other hand, an administrator might query

```
GET /stafffields/Fred
```

and receive

```
<person>
  <idurl> http://XYZ/1234 </idurl>
  <nameurl> http://XYZ/1234/name </nameurl>
  <phoneurl> http://XYZ/1234/phone </phoneurl>
</person>
```

The administrator could then use these urls to perform PUT operations to update the values such as

```
PUT http://XYZ/1234/phone?value=9876543
```

XML and representations

In the examples above, I returned XML documents. Such a format is not specified by REST: all that REST requires is that *something* be returned.

The book "RESTful Web Services" returns HTML documents in its examples. While simple for browsers to show, for machine processing the HTML has to be "scraped" as it contains little semantic content. I prefer to return an XML document with an XSLT stylesheet if the document is also intended for human use.

Higher levels

Where is the equivalent of WSDL? Of UDDI? Of the rest of the WS-* stack? By and large, these do not exist. There have been studies on a RESTservices description language, and by-and-large these have avoided the mistakes of WSDL (Newmarch 2003). But none of these have become standards as yet. Since REST services have URIs, ordinary search engines like Google can pick them up, and there are an increasing number of REST-based services with published URIs and how to access them. Security is handled by HTTPS in general, although any encryption or authentication technique could be used if needed, including all of those supported by HTTP.

Conclusion

In comparison to the WS* stack, REST offers a simple style of accessing resources that works with the Web rather than against it. Given a choice, programmers seem to prefer using RESTstyle APIs rather than SOAP APIs, and the style is supported by an increasing number of vendors and servers.

REST says nothing about requirements such as service orchestration, and currently there are no REST standards that would make it easy to design a framework to satisfy such requirements. For a large number of SOA systems, REST is a good architectural solution, and there are enough examples in the lierature to design systems following these principles.

References:

Roy Thomas Fielding (2000). Architectural Styles and the Design of Network-based Software Architectures, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, last accessed September, 2008

W3C Technical reports (2008). http://www.w3.org/TR/ last accessed August, 2008

SOAP Specification version 1.2 (W3C) (2007). http://www.w3.org/TR/soap/, last accessed August, 2008

W3C (2008) HTTP – Hypertext Transfer Protocol revised version 1.1 , lhttp://www.w3.org/Protocols/ast accessed August, 2008

Jan Newmarch (2003). A Critique of Web Services, http://jan.newmarch.name/webservices/critique.pdf
WWW Consortium *Web Services Architecture* http://www.w3.org/TR/2003/WD-ws-arch-20030514

Jan Newmarch (2005) UPnP Services and Jini Clients , ISNG 2005, Las Vegas

W3C, *SOAP Version 1.2 Part 0: Primer* http://www.w3.org/TR/2002/WD-soap12-part0-20020626/
WSDL 1.0 Specification, http://http://www.ibm.com/developerworks/web/library/w-wsdl.html