MEMBERSHIP
ARTICLES
ALL ARTICLES
WHITE PAPERS
ROUND TABLES
PRESENTATIONS
NEWS CLIPPINGS
EVENTS
TRAINING
WORKSHOPS
CONSULTANT NETWORK
SOLUTION LOCATOR
SEARCH

OTHER TOPICS
BPM
BIZ DECISION MGMT
BIZ ARCHITECTURE
ORG. PERFORMANCE
INNOVATION
GOVERNMENT

SOLUTION LOCATOR

Expedite your research.
Find specific SOA solutions and request information.

ORACLE

ORACLE

SOA WATCH COLUMN

ARTICLES

RESTful Web Services Part I: Concepts and Design

By: Jeremy Deane, Technical Architect, Collaborative Consulting
Wednesday, October 15, 2008

An increasing industry backlash to the mounting complexity of the WS-* SOAP specifications is resulting in many organizations considering RESTful Web Services. In fact, a recent poll by *Information Week* found that 58% of IT professionals believed that SOA introduced more complexity and resulted in cost overruns. Some industry pundits, such as Tim Bray echo this sentiment declaring WS-* an embarrassing failure.

So why turn to REST? REST or Representational State Transfer is based on a small set of widely-accepted standards such as Hypertext Transfer Protocol (HTTP), Uniform Resource Identifier (URI), and Extensible Markup Language (XML). REST requires far fewer development steps, toolkits, and execution engines than SOAP. Many organizations have implemented mission critical systems using this simple architectural style including Amazon S3, Flickr, and Overstock.

In this article I will cover some of the key concepts behind REST and walk through the process of designing RESTful web services. In a subsequent article I will cover implementing and testing RESTful web services. Both articles will use a fictional facilities management web application, as an example, where a series of RESTful web services for managing facilities (e.g. adding rooms to a building) will be exposed.

A RESTful web service provides access to a resource (e.g. building, room), identified by URI[1] , using HTTP. A resource is an abstraction of information. For instance, "Room 12B" is an abstraction with representations of that resource realized as a web page, an XML document, or a PDF. Consequently, a resource requires an identifier or URI to identify specific representations.

The HTTP methods, PUT, GET, POST, and DELETE define a uniform interface for accessing resources (i.e. **C**reate, **R**etrieve, **U**pdate, **D**elete or CRUD). HTTP is stateless request-response *application protocol*. Request and response messages are comprised of a command (method), a header, and a body. The interactions can be secured at the transport layer using the Secure Sockets Layer (SSL) protocol while the messages can be secured using encryption and a digital signature.

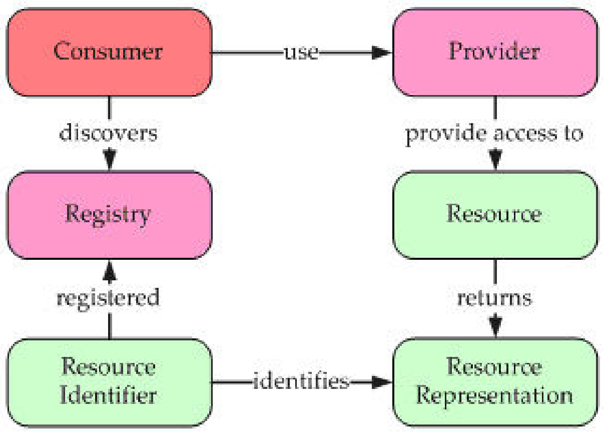


Figure 1: REST Conceptual Model

A resource representation is a self-descriptive, immutable physical snapshot of a resource. The most common resource representation is XML but other popular formats include JSON and XHTML. A representation may contain links to other resources (a.k.a. hypermedia). The use of *hypermedia* within representations allows consumers and providers to evolve independently over time.

A resource representation and its links correspond to a snapshot of the *application state*. Each interaction results in an updated application state; however, a resource's state only changes in the case of a PUT, POST or DELETE. Furthermore, these interactions are said to be *idempotent*, meaning that duplicate consumer requests to create, update, or delete a resource are processed once and only once.

RESTful web services are secured by implementing authentication, authorization and auditing. *Authentication* is assurance of a consumer's identity whereas *authorization* verifies that an identified consumer has the appropriate privileges. A privilege is comprised of a method and URI (e.g. POST ../building/{building-id}). Finally, *auditing* entails logging any action that updates the state of a given resource.

The first step in designing a RESTful web service is to answer several questions about your domain:

1. What are your entities (resources)?
2. How do your consumers want the data formatted (resource representations)?
3. Are there relationships between the resources (links)?
4. What identifies each entity (URI)?

In our facilities management web application, the resources include buildings and rooms.

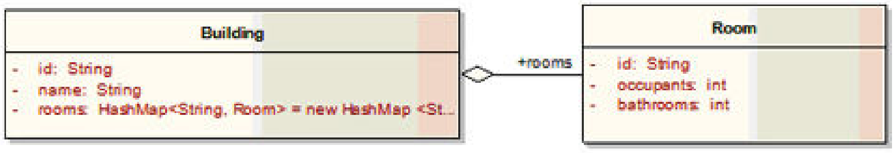
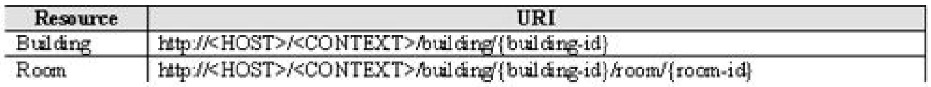


Figure 2: Facilities Management Domain Model

The consumers require representations of that data formatted in XML and buildings contain links to rooms. Finally, the resources are identified by the following URIs:



Resource	URI
Building	http://<HOST>/<CONTEXT>/building/{building-id}
Room	http://<HOST>/<CONTEXT>/building/{building-id}/room/{room-id}

While there is wide agreement on the underlying architectural concepts of REST, to what degree a web service is "RESTful" is open for debate. URI design is a main area of contention. In general, a URI should not contain verbs or resource identifying parameters.

Incorrect …/building/{building-id}?town=Newton

Correct …/{town}/building/{building-id}

However, this dogmatic approach may limit the types of RESTful web services you can implement (e.g. search can be problematic without additional parameters). Other aspects of URI design include whether to incorporate hierarchical patterns and the granularity of the URI itself. Of course the granularity of a URI is also constrained by the granularity of the corresponding resource. Regardless of your approach, the key is to make the URI descriptive and scalable.

Another design consideration is the use of HTTP application status codes[2] . For example upon successfully creating a resource, the HTTP response could contain the code 201 *(created)* or a generic 200 *(ok)*; the codes could essentially be ignored and an agreed upon status could be included into the message itself. The former approach is not recommended since it results in tighter coupling between consumer and provider.

RESTful web services reduce integration complexity, decrease time to market, and result in low-friction systems. RESTful web services allow a development team to spend more time integrating systems than reading through some of the 1300 pages of WS-* specifications. In the next article, I will demonstrate how to implement web services using Restlet, a REST framework for Java, and how to test those services using Apache JMeter, a desktop testing tool.

[1] implemented as a Universal Resource Locator (URL)

[2] HTTP Application Status Codes: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Back to Articles, including SOA, BPM & BA

or join today!

READ MORE ON SOAINSTITUTE.ORG

Featured Presentation:

Patterns for implementing SOA solutions
Featuring: Boris Lublinsky, Co-Author of "SOA Applied: Architecture and Design Strategy"

In this presentation we will discuss practical approaches to building enterprise solutions using existing enterprise business services. We will start by defining the overall architecture of the...

more

Featured White Paper:

SOA Architecture Considerations
Courtesy of: Keshav Deshpande, Principal Consultant, Keane, Inc.

An important part of analyzing necessary capabilities for a future Service Oriented Architecture (SOA) and Enterprise Interoperability environment is to envision, explore and lay out the core...

more