ROA vs. Big Web Services

Kenneth M. Anderson
University of Colorado, Boulder
CSCI 7818 — Lecture 10 — 10/29/2008

Agenda

- Yahoo Pipes
- Atom Publishing Protocol
- Discussion of Chapter 10 of Textbook

- But first…
  - a discussion of some pointers sent to me by Steven
  - and one of my own on “the first servers”

Chapter 10: A comparison of ROA and BWS

- Chapter 10 spends time examining Big Web Services (BWS, aka WS-*) and how they compare with REST and ROA
    - The chapter does not contain detailed coverage of BWS technologies but covers enough to examine how the philosophies line up
- Starts with Web comparison
    - Web is based on resources; BWS do not expose resources
        - To implement RPC on top of the Web goes against its grain
    - Web is based on URIs and links; BWS: one URI, no links
    - Web is based on HTTP; BWS hardly uses HTTP's features
- As a result, BWS are not addressable, cacheable, well connected, and they don't respond to a uniform interface; understanding one does not mean you'll understand the next, and they tend to have interoperability problems

What problems are BWSs trying to Solve?

- The authors describe a typical example application that BWSs try to solve
  - Typical Travel Agent Scenario
    - Book flight, rental car, and hotel
    - Requires coordination with multiple external entities
    - Time-constrained: Airline may be willing to hold “seat 24C” for 5 mins.
- Thus BWSs are trying to solve:
  - the design of process-oriented, brokered distributed services
- The authors assert that since the ROA is turing-complete, it can be used to solve these problems as well
  - it would require careful resource design, with some resources having limited value: such as the “hold search 24C for 5 mins.” resource

SOAP

- SOAP as described by Richardson and Ruby
  - “You can take any XML document (…), wrap it in two little XML elements, and you have a valid SOAP document. For best results, though, the document’s root element should be in a namespace.”
  - The key benefit of SOAP is transport independence
    - since body and headers (“stickers on the envelope”) are all contained within the SOAP envelope, any transport can be used to send SOAP messages
      - in practice, though, only HTTP is used
- Nothing too objectionable here: “SOAP is mainly infamous for the technologies built on top of it.”

The Resource-Oriented Alternative

- The difference between the RPC-based approach facilitated by SOAP and the REST-based approach is explained by analogy with OO and structured programming languages
- In the latter
  - my_function(object, argument)
- In the former
  - object->my_method(argument)
- To convert, start pulling resources out from behind the single URI of BWS
  - You'll find groups of resources that "behave" the same enabling a uniform interface: analogous to polymorphism in OO languages

WSDL

- The authors work through the simplest possible example of using WSDL
  - For a service that lives at http://www.soapware.org/weblogsCom
    - This service exposes one operation “ping”
      - ping takes two strings and returns a pingResult structure
        - The pingResult structure consists of a boolean and a string
- Lets view what it takes to define this service in WSDL

First, define the `pingResult` Type

```
<types>
  <s:schema targetNamespace="uri:weblogscom">
    <s:complexType name="pingResult">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1"
        name="flerror" type="s:boolean" />
        <s:element minOccurs="1" maxOccurs="1"
        name="message" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:schema>
</types>
```

Second, define the ping messages

```
<message name="pingRequest">
  <part name="weblogname" type="s:string" />
  <part name="weblogurl" type="s:string" />
</message>
```

```
<message name="pingResponse">
  <part name="result" type="tns:pingResult" />
</message>
```

Third, define the port type

```
<portType name="pingPort">
  <operation name="ping">
    <input message="tns:pingRequest" />
    <output message="tns:pingResponse" />
  </operation>
</portType>
```

The definition is still abstract. It could be implemented in a number of ways. So, now we need to specify the concrete information.

Fourth, bind the portType to an implementation

```
<binding name="pingSOAP" type="tns:pingPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="ping">
        <soap:operation soapAction="/weblogUpdates" style="rpc" />
        <input>
            <soap:body use="encoded" namespace="uri:weblogscom" encodingStyle="http://
            schemas.xmlsoap.org/soap/encoding/" />
        </input>
        <output>
            <soap:body use="encoded" namespace="uri:weblogscom" encodingStyle="http://
            schemas.xmlsoap.org/soap/encoding/" />
        </output>
    </operation>
</binding>
```

Now we must bind this “binding” to a service that provides an enpoint URI

Fifth, define the service

```
<service name="weblogscom">
    <document>For a complete description of this service...</document>
    <port name="pingPort" binding="pingSoap">
        <soap:address location="http://rpc.weblogs.com:80/" />
    </port>
</service>
```

WSDL Breakdown

- That's a lot of work to define a single operation that accepts two strings and returns a boolean and a string!
  - WSDL makes no simplifying assumptions, everything has to be specified every time you write a new spec
    - As a result of this complexity, tools become the real story and you become dependent on your tools
- The problem from the authors perspective is that
  - you move further and further away from the Web
  - the generated interfaces tend to be brittle
    - different tools generate slightly different WSDL files leading to interoperability problems
- None of these complexities help solve the travel broker problem, and these complexities attack other desirable characteristics (simplicity/scalability)

Resource-Oriented Alternative

- WSDL serves two main purposes in BWSs
  - It describes the interface the service exposes
  - It describes the representation formats
- In resource-oriented services, these functions are often unnecessary or can be handled with much simpler standards
  - The uniform interface solves the first, using pre-defined formats, such as Atom or HTML can solve the latter
- From REST perspective, the problem with WSDL is that it encourages the design of single endpoint services with all functionality exposed via overloaded POST operations
  - It also has no provisions for defining hypertext links (as its focus is on operations, not resources)

UDDI

- UDDI is the “yellow pages” for WSDL
  - A way for clients to look up a service that fits there needs
- Surprisingly, UDDI is even MORE complex than WSDL (as we’ve seen)
- The vision of UDDI was one of multiple registries
  - a fully-replicated Internet-scale registry for businesses
  - and a private registry behind the firewall of any company that wanted to host one
- The latter model has occurred since single companies can devote resources to ensure quality control on the information contained in the registry
  - A public UDDI registry maintained by IBM/Microsoft shut down in 2006 after containing entries for 50K business, unfortunately quality control on this information was low and the service did not get adopted

Resource-Oriented Alternative

- The author's concede that there is no silver bullet to this problem
  - An automated system that helps people find hotels has a built-in economic incentive for hotel chains to game the system
    - Take a look at the behavior around the iTunes App Store
      - http://www.dragthing.com/blog/?p=30
      - http://hothardware.com/News/iPhone-App-Developers-Gaming-The-System/
      - http://www.betanews.com/article/Some_iPhone_app_devs_game_the_system_for_higher_placement/1216051901
- For REST, the closest equivalent to UDDI are search engines
  - They help (human) clients find the resources they are looking for
  - spammers can (and do) game this system however

What about X?

- The rest of Chapter 10 takes a “What about X?” approach where X is one of
  - security
  - reliable messaging
  - transactions
  - BPEL, ESB, and SOA
- In each case, there are more specifications on the BWSs side
  - The books recommendation typically follows the form of
    - Make sure you really need this
    - If so, attempt to port a BWS approach to HTTP headers to gain some of the benefits

Coming Up Next

- Next week: Introduction to Web 2.0
    - Any volunteers for some initial Web 2.0 presentations?
        - Social Networking Sites: Ning, Facebook, MySpace
        - Web 2.0 News Sites: newsvine.com
        - AJAX
        - Javascript Toolkits for Rich Application Development
        - Google App Engine, Amazon’s EC2, Microsoft Windows Azure
        - etc.