RESTful Web Service Composition: putting BPEL to REST

Cesare Pautasso
Faculty of Informatics
University of Lugano (USI), Switzerland

http://www.pautasso.info
University of Lugano, Switzerland

- Faculty of Informatics (Opened 2004)
- 15 Professors
- 63 PhD + Post-doc

- Excellence in Research
- Innovative Teaching
My Research Interests

- Web 2.0
- Mashups
- Business Process Management
- Scientific Workflow Management
- Autonomic Computing
- Service Oriented Architectures
- Software Composition

©2008 Cesare Pautasso
RESTful Web Service Composition: putting BPEL to REST

Cesare Pautasso
Faculty of Informatics
University of Lugano (USI), Switzerland
http://www.pautasso.info
Motivation – Why BPEL for REST?

"The WS-BPEL process model is layered on top of the service model defined by WSDL 1.1. [...] Both the process and its partners are exposed as WSDL services"

[WS-BPEL 2.0 Standard, Section 3]
RESTful Web Services APIs...

...do not use WSDL 1.1
The Goal

- Compose RESTful Web Services
- Compose WSDL Web Services
- Use Business Process Modeling Languages

One Solution: BPEL for REST

- Extend BPEL to support RESTful Web Services
Web Sites (1992)

- Web Browser
- HTML
- HTTP
- Web Server

WS-* Web Services (2000)

- Client
- SOAP
- XML
- (HTTP)
- WSDL
- Server
RESTful Web Services (2007)

WS-* Web Services (2000)
Application Integration Styles vs. REST

- Stream
- Remote Procedure Call
- Message Bus Events
- File Transfer
- Shared Data
- REpresentational State Transfer
REST in one slide

- Web Services expose their data and functionality through resources identified by URI
- **Uniform Interface** Principle: Clients interact with the state of resources through 4 verbs: GET (read), POST (create), PUT (update), DELETE
- **Multiple representations** for the same resource
- **Hyperlinks** model resource relationships and valid state transitions
The Challenges

- Resource addressing through URI
  - *Interact with dynamic, variable set of URI*

- Uniform Interface (GET, POST, PUT, DELETE)
  - *Make the 4 verbs explicit in the composition language*

- Multiple resource representations
  - *No static message types*
  - *Negotiate with clients the most appropriate representation*

- Hyperlinks
  - *Implement state transition logic of a resource*
  - *Generate new URIs dynamically as processes runs*
The Hack – Without BPEL for REST

WSDL 2.0 HTTP Binding can wrap RESTful Web Services

(WS-BPEL 2.0 does not support WSDL 2.0)
Native support for direct invocation of RESTful Web services

REST concepts first-class language constructs
Agenda – BPEL for REST

- Motivation
- BPEL for REST Extensions
  - Invoking RESTful Web Services
  - Publishing RESTful Web Services
- Example
- Outlook
BPEL for REST - invocation primitives

<put/>
<get/>
<post/>
<delete>

Resource

PUT
GET
POST
DELETE

Web Service

<i>> invoke<\i></i>

<i>> invoke<\i></i>

Web Service

<i>> receive<\i></i>

<i>> reply<\i></i>

<i>> receive<\i></i>
Invoking RESTful Web Services

- 4 new activities (get, post, put, delete)

```xml
<get uri="" response="">
  <header name="">*value</header>
  <catch code="" faultName=""?>*... </catch>
  <catchAll>?... </catchAll>
</get>

<post uri="" request="" response="">... </post>
```
Publishing RESTful Web Services - I

- 4 new request handlers
  (onGet, onPut, onDelete, onPost)
- 1 new “scope” (resource)

<resource uri=""/>

<variable>*

<onGet>? . . . </onGet>

<onPut>? . . . </onPut>

<onDelete>? . . . </onDelete>
Publishing RESTful Web Services - II

- 1 new activity (response)

```xml
<response code=""/>

<header name="">*value</header>

payload

</response>
```
Example

- Loan Application Service Process

Client

- POST /loan
- GET /loan/choice
- PUT /loan/choice
- GET /loan
- DELETE /loan

Loan Service (BPEL)

- GET /rate
- POST $accept

Bank Services
1. State of the Resource

- Declarative construct to publish a resource URI

```xml
<process name="LoanApplication">
  <resource uri="loan">
    <!-- State variables of the resource -->
    <variable name="name"/>
    <variable name="amount"/>
    <variable name="rate"/>
    <variable name="bank"/>
    <variable name="start_date"/>
    <variable name="end_date"/>
  </resource>
</process>
```
2. Handle POST request - Preconditions

- Create a new loan application resources only if...

```xml
<onPost />
  <if>
    <condition>$request.amount &gt; 100000</condition>
    <then>
      <respond code="400">
        Requested amount too large
      </respond>
      <exit/>
    </then>
  </if>
```

3. Handle POST request - Initialization

- Store initial loan application resource state

```xml
<else>
<sequence>
<assign>
    name = $request.name;
    amount = $request.amount;
    start_date = $request.start_date;
</assign>
<respond code="201">
    <header name="Location">/loan/$name</header>
    Processing loan application...
</respond>
```
4. Invoke RESTful Web Services

- Get rates from banks

```xml
<scope>
  <variable name="ubs_response"/>
  <variable name="cs_response"/>
  <variable name="url_accept"/>
  <variable name="accept_response"/>
</scope>

<flow>
  <get>
    uri="http://www.ubs.ch/rate?chf=$amount&from=$start_date"
    response="ubs_response">
  </get>

  <get>
    uri="http://www.cs.ch/rates?amount=$amount&start=$start_date"
    response="cs_response">
  </get>
</flow>
```
5. Let Client Choose – GET Handler

- Return rates offered by the banks

```xml
<while>
<condition>TRUE</condition>
<resource uri="choice">
  <onGet>
    <respond code="200">
      <header name="Content-Type">application/json</header>
      [ { bank: "cs",
          rate: "$cs_response.rate",
          end_date: "$cs_response.until" },
       { bank: "ubs",
          rate: "$ubs_response.rate",
          end_date: "$ubs_response.end" } ]
    </respond>
  </onGet>
</resource>
```
6. Let Client Choose – POST Handler

- Store the client choice and continue

```xml
<onPost>
<sequence>
  <assign>bank = $request.choice;</assign>
  <if>
    <condition>bank == "cs"</condition>
    <then>
      <assign>rate = $cs_response.rate;
          end_date = $cs_response.until;
          url_accept = $cs_response.accept</assign>
    </then>
    <else>
      <assign>rate = $ubs_response.rate;
          end_date = $ubs_response.end;
          url_accept = $ubs_response.accept</assign>
    </else>
  </if>
  <respond code="200"/>
</sequence>
</onPost>
```

7. Inform Bank

- Accept the loan offered by the chosen bank

```xml
<post uri="$url_accept" request="$name"
    response="accept_response">
  <scope/>
  <sequence/>
  <else/>
  <if/>
</onPost>
```
8. Let Client Choose – DELETE Handler

- Cancel the loan application or cancel the loan

```xml
<onDelete>
  <if>
    <condition>bank == null</condition>
    <then>
      <respond code="200"/>
      <exit/>
    </then>
    <else>
      <!-- Start the loan cancellation process -->
      <invoke...>
    </else>
  </if>
</onDelete>
```
BPEL for REST Extensions - Summary

- Publishing and Invoking RESTful Web Services
Mapping the State of Resources and Processes

- Correlation
- Lifecycle
- Visibility
- Access Control
Reference Architecture

BPEL for REST Engine

Front-End

HTTP Server
Resource Publisher
Servlet

BPEL Engine
Process Execution
Persistent Execution State

Back-End

Invocation Activities
RESTful
HTTP Client
Conclusion

- Business Process Modeling Languages have been applied with success to compose “traditional” WS-* Web Services (BPM = SOA + BPEL)
- Business Process Modeling Languages should also be applied to compose RESTful Web Services
- BPEL for REST is a lightweight WS-BPEL extension for composing both kinds of services
More information


PhD positions available!

Prof. Cesare Pautasso
University of Lugano, Switzerland
c.pautasso@ieee.org
http://www.pautasso.info