RESTful Web service composition with BPEL for REST

Cesare Pautasso
Data & Knowledge Engineering (2009)

2010-05-04

Seul-Ki Lee
Contents

- Introduction
- Background
- Design principles of RESTful Web service
- BPEL for REST extension
- Implementation
- Reference architecture
- Related work
- Conclusion
Web Service

- Describes web-based applications over the web
  - Helpful to the distributed development environments
  - Web is enable to on-the-fly programming (or live coding)
    - Easy to evolve and enhance the applications
- Separates into interface and implementation
  - Do not concern how these services will execute client’s requests
  - More flexible to integrate and modularize the applications
Traditional Web service

- SOAP Web service

- Service providers describe their services in the WSDL
- Service requesters find required services from UDDI
  - Bind services provided by service provider
- However, SOAP is too heavy to send/receive messages
  - Encoding and decoding are needed

WSDL: Web Service Description Language
UDDI: Universal Description, Discovery and Integration
SOAP: Simple Object Access Protocol
RESTful Web service

REpresentational State Transfer (REST)
- Directly maps to the primitives as resource by using the interface without encoding
  - E.g., GET, POST, PUT, DELETE
  - Providing and consuming services is simpler than SOAP Web service
Service workflow language

- Description for processing services by the order
- No standard for describing RESTful Web services
  - Current Web service description languages are only considering SOAP Web service
In this paper

- Suggest extended workflow language for RESTful Web service
  - Composes RESTful Web services by directly invoking related services
  - Publishes processes as RESTful Web services
Business Process Execution Language (BPEL)

- Describes business logic between service interactions
  - Web service description languages do not contain the execution order of services
  - Gain broad acceptance in industry and research
    - De facto standard of business process language
- Publishes composite new Web services
  - External Web services are interacted by partners

Workflow language (BPEL)
BPEL structure

BPEL: Business Process Execution Language
WS: Web service
Resource addressing through URI
- Needs of late(dynamic) binding
  - URIs are not always known in advance
  - Require some form of dynamic discovery
- Needs of state transition specification
  - How the state of arbitrary resources may evolve

HTTP related uniform interface
- GET – retrieve the current state of a resource
- PUT – transfer a new state onto a resource
- DELETE – delete an existing resource
- POST – create subordinate resources (instantiation)
Self-descriptive messages for the flexibility

- Services provide multiple representations for a given resource
  - Select most suitable format
- Uses meta-data to control HTTP properties
  - HTTP request and response headers are needed

Generate new resource URIs

- Consider the valid future states
  - Embedded in the representation of resource as hyperlinks
  - Extract URIs from response messages
Composing RESTful Web service

Add four activities to invoke a RESTful Web service

- Specify the address of the target resource

\[
\text{<get uri="" response="" response_headers=""?>}
\text{<header name="">*value</header>}
\text{<catch code="">*...</catch>}
\text{<catchAll>?...</catchAll>}
\text{</get>}
\]

- Store the status code for the response

\[
\text{<post uri="" request="" response="" response_headers=""?>}
\text{...}
\text{</post>}
\]

- Store data for the request and response

\[
\text{<put uri="" request="" response="" response_headers=""?>}
\text{...}
\text{▲ Store data for the request and response}
\text{<get> <delete> do not have request, <put> <delete> have optional response )}
\text{</put>}
\]

\[
\text{<delete uri="" response="" response_headers=""?>}
\text{...}
\text{</delete>}
\]
Publishing RESTful Web service

- Publishing processes as RESTful Web services

```xml
<resource uri="">
  <variable>*
    <onGet>? ... </onGet>
    <onPut>? ... </onPut>
    <onDelete>? ... </onDelete>
    <onPost isolated="false"><? ... </onPost>
  </variable>
</resource>
```

- Declare the state of the resource found at a given uri

- A set of request handlers trigger when the processes receives the corresponding HTTP request

- Concurrent activation is enabled

```xml
<respond code=""/>
  <header name="">*value</header>
  payload
</respond>
```

- Results

- More than one respond attributes are possible for multiple data

ⓒ KAIST SE LAB 2010
Implementation (1/4)

**e-Commerce scenario**

1. Clients browse product catalogs and request price quotes
2. Clients place an order
3. Clients add or remove items
4. Clients update the order with the payment information
5. Gathers corresponding products
6. Ships the products
e-Commerce scenario (cont’d)

Create new order

```xml
<process name="ShopService">
  <resource uri="order">
    <!-- Assume variables are declared -->
    <onPost>
      <!-- POST /order request handler -->
      <if><condition>$request.customer == null</condition>
        <sequence>
          <respond code="400">
            Cannot create a new order: missing customer information
          </respond>
        </sequence>
      </if>
      <else>
        <sequence>
          <assign>
            oid = new ID("/order");
            customer = $request.customer;
            state = 'Empty';
            items = [];
          </assign>
          <respond code="201">
            <header name="Location">$oid</header>
            Order $oid has been created
          </respond>
        </sequence>
      </else>
    </onPost>
  </resource>
</process>
```
e-Commerce scenario (cont’d)

- Add and remove item

```xml
<resource uri="item">
  <variable name="iid"/>
  <variable name="product"/>
  <variable name="quantity"/>
  <variable name="price"/>
</resource>

<!-- POST /order/item request handler -->

```

- Save <get> result in the variable

```xml
<onPost>
  <sequence>
    <variable name="quote_response"/>
    <get url="$request.quote" response="quote_response"/>
    <assign>
      iid = new ID("/order/item");
      product = quote_response.product;
      price = quote_response.price;
      quantity = quote_response.quantity;
      state = 'Ready';
      items[$iid] = {iid: $iid, product: $product, price: $price, quantity: $quantity};
    </assign>
  </sequence>
</onPost>

- Set selected item to null

```
e-Commerce scenario (cont’d)

Confirm payment information

```
<resource uri="payment">
  <!-- POST /order/payment request handler -->
  <onPost>
    <sequence>
      <variable name="payment_response"/>
      <assign>
        amount = computeTotalOrderAmount($items);
        payment = $request.payment;
        payment.amount = $amount;
      </assign>
      <post url="https://www.visa.com/payment" request="payment" response="payment_response">
        <catch code="200">
          <sequence>
            ▼ Change state: cannot modify order anymore
            state = "Confirmed";
            payment.id = payment_response.confirmation;
          </assign>
          <respond code="200">
            Payment has been accepted: $payment_response
          </respond>
        </catch>
        <catchAll>
          <respond code="500">
            Could not complete payment: $payment_response
          </respond>
        </catchAll>
      </post>
    </sequence>
  </onPost>
</resource>
```
Integrated BPEL engine

- Front-End
  - Handles HTTP requests from clients by routing them to the corresponding request handler

- Back-End
  - Invoke and bind a RESTful service down to an HTTP request-response interaction
**Related work**

- **Proposed approaches**
  - Modification of existing BPEL

<table>
<thead>
<tr>
<th>Proposed approach</th>
<th>H.Overdick</th>
<th>Bite</th>
<th>BPEL for REST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Translated BPEL with internal state of resource</td>
<td>Simplified variant of BPEL with a reduced set of activities</td>
<td>Extended BPEL with a set of resource-oriented activities</td>
</tr>
<tr>
<td>Activities support</td>
<td>All activities except invocation and BPEL process as resource</td>
<td>All activities except PUT</td>
<td>All activities</td>
</tr>
<tr>
<td>Publishing</td>
<td>Direct</td>
<td>Indirect</td>
<td>Direct</td>
</tr>
<tr>
<td>Resource declaration</td>
<td>Dynamic</td>
<td>Static</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>
Conclusion

❖ Contribution

▪ Propose an extended BPEL for REST
  • To support the composition of RESTful Web services
  • Add BPEL activities for GET, POST, PUT and DELETE

❖ Future work

▪ Publishing new resource by processes or creating new process by resource
  • Process can create specific resource instance by HTTP Cookie or URI rewriting method
Discussion

❖ Pros

▪ Propose an extended workflow language
  • Without loss of RESTful Web service’s advantages
  • With original traditional Web service’s form

❖ Cons

▪ Do not show the composition result between traditional Web service and RESTful Web service
▪ Missing implementation is existing
Q & A

THANK YOU
## Separation of roles

<table>
<thead>
<tr>
<th></th>
<th>SOAP Web service</th>
<th>REST Web service</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit</strong></td>
<td>Process</td>
<td>Resource</td>
</tr>
<tr>
<td><strong>Applicable area</strong></td>
<td>Transaction processing</td>
<td>Data and UI (User Interface) processing</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Robust syntax checking</td>
<td>None</td>
</tr>
<tr>
<td><strong>Context management</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Standard</strong></td>
<td>WSDL</td>
<td>None</td>
</tr>
<tr>
<td><strong>Limitation</strong></td>
<td>Complex usage method</td>
<td>Absence of standard</td>
</tr>
<tr>
<td></td>
<td>Heavy protocol</td>
<td>Difficulty of management</td>
</tr>
</tbody>
</table>
In my approach

- Finds **optimized** composition method for complex Web services with previous BPEL
  - No distinction between SOAP web service and RESTful Web service are required
  - Two registries can be same
Current Issues

- Development environment
  - The number of distributed developments is increasing
  - Hard to integrate the enterprise applications
  - Requirements are more complex
  - On-the-fly programming (or live coding) are required

Cost-effective solutions to evolve and enhance are needed
Service Oriented Architecture (1/2)

Service oriented architecture

Service

- A set of related **software functionalities**, together with the policies that should control its usage.
- “A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraint and polices as specified by the service description” – OASIS

Web service

- Services delivered over the web using technologies
  - e.g., XML, WSDL, SOAP and UDDI
Service oriented architecture (cont’d)

- **Architecture style**
  - Builds software application that use services available in a network such as the web

- **Separation of the service interface and implementation**
  - Clients are not concerned with how these services will execute their requests
  - Services are self-contained and loosely coupled

- **Dynamic discovery**
  - Composite services can be built from aggregates of other services
BPEL notation

- **Request-response structure**
  - `<receive>`: request message from an external partner
  - `<reply>`: response message corresponding to `<receive>`
  - `<invoke>`: call a Web service provided by a partner

- **Lack of information for data flow**
  - Data is stored in shared variables
    - Accessed by `<assign>` activity

- **Structured control flow**
  - `<sequence>`: nesting structured activities
  - `<flow>`: parallel control flow
  - `<if>`: conditional branches in the control flow
  - `<links>`: connecting other activities in a `<flow>` activity