RESTful SOA and XML

W3C Workshop on Data and Services Integration

Cornelia Davis
Sr. Technologist
EMC CTO Office
October 2011
Overview

• Examples of data and services integration challenges (at EMC)
  – Using many different management products collect information about data center entities to assess the state of that data center
    • And if the logical “data center” is, in fact, the cloud?...
  – Search over a variety of content repositories
    • Again, within a data center or in the cloud
  – ...

• The EMC CTO office is, in fact, working on developing that “coherent platform” for (both!) data and services integration

• My assertion – the following play a significant role:
  – REST over HTTP
  – XML
Services Integration: REST with HTTP

Why REST?

- In order to get some standardization and uptake, the principles need to be easy
- The environment is highly distributed and potentially very large – that is, CLOUD
- Product groups are all building RESTful interfaces
- Resource focus
  - With media-type handling
  - With hyperlinking
  - With caching
REST Principles

• REST is an **architectural style** that depends upon:
  – Identification and addressability of resources
    – All interesting bits of information are identified with URIs and are usually accessed via URL
• The uniform interface
  – Interaction with resources through a standardized set of operations, with well understood semantics
• Manipulation of resources through representations
  – Media types
• Hypermedia as the engine of application state
  – Hyperlink your resources
And A Few More Details...

• We talk about:
  – The need for stable identifiers and accessors
    • In support of HATEOAS, caching
  – The need for a mechanism for determining of resource state has changed – eTags
    • In support of caching
  – Not finding a resource is perfectly valid
    • In support of loose coupling
  – And more...

How we address these needs mustn’t be “disconnected from the general Web usage”
  – That is, I want not only REST, but I want to use HTTP.

• The REST architectural style
• HTTP as the protocol
# How Well Do Existing REST Frameworks Do?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Java FWs (alone)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Named Resources</td>
<td>●</td>
<td>URI template support - @RequestMapping(&quot;/patients/{pid}&quot;))</td>
</tr>
<tr>
<td>Define Uniform Interface</td>
<td>●</td>
<td>@RequestMapping (method = RequestMethod.GET) (+ PUT, POST, DELETE, PATCH, etc.)</td>
</tr>
<tr>
<td>Handle media types</td>
<td></td>
<td>Frameworks do allow bad practices; most don’t fully support features like media type parameters</td>
</tr>
<tr>
<td>Link to other resources</td>
<td>○</td>
<td>nothing</td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td>Generally happens with a bunch of java code.</td>
</tr>
</tbody>
</table>
RESTful Service Written in Java

Spring
- Parses URL, instantiates the right annotated POJO and calls it.

Marshallers/Unmarshallers
- Serializes model object into XML or JSON String

... then uses...

Service POJO
- Accesses persistence, instantiates model object and populates with data

Patient

Results
...now if we also want Atom representations

Spring
Parses URL, instantiates the right annotated POJO and calls it.
... then uses ...

Service POJO
Accesses persistence, instantiates model object and populates with data

Abdera
Serializes FOM object into XML or JSON String

Marshallers/Unmarshallers
Serializes model object into XML or JSON String

FOM-Pat
Results

String
RESTful Service – *XML as the Dial Tone*

Spring
*Parses URL, instantiates the right annotated POJO and calls it.*

Service POJO
*Accesses persistence, generates XML model and returns*
XML Technologies are Well Suited to...
(And the “XML stack has become ubiquitous”)

• Transformations to media types
• Hyperlink insertion

...<xsl:template match="p:Patient" mode="inserthere">
  <atom:link rel="self">
    <xsl:attribute name="href"><xsl:value-of select="$baseURL" /></xsl:attribute>
  </atom:link>
  <atom:link rel="prescriptions">
    <xsl:attribute name="href"><xsl:value-of select="concat($baseURL,'/activeprescriptions')" /></xsl:attribute>
  </atom:link>
  <atom:link rel="episodes">
    <xsl:attribute name="href"><xsl:value-of select="concat($baseURL,'/careepisodes')" /></xsl:attribute>
  </atom:link>
  <atom:link rel="up">
    <xsl:attribute name="href">
      <xsl:value-of select="functx:substring-before-last($baseURL,'/')" /></xsl:attribute>
  </atom:link>
</xsl:template>
...But are Lacking In Some Things

- Naming
- Uniform interface
- HTTP “libraries”

Things that the Java-based RESTful service frameworks are good at
XML-Centric Implementation

- POJO for RESTful Service "declaration"
- XQuery Logical (resource) to physical mediation
- XSLT for hyperlink insertion and media type support
- XML Database

XProc Pipeline
core services implementation

XProc pipeline steps
“Declaring” Services

In the POJO:  

Encapsulate operation pipeline and design time bindings

```java
@Controller
@RequestMapping("/patients")
public class Patients {
    private static XMLProcessingContext m_getPatientsProcessing = null;
    private static XMLProcessingContext m_addPatientProcessing = null;

    public void setAddPatientProcessing(XMLProcessingContext val) {
        m_addPatientProcessing = val;
    }
    ...
    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.CREATED)
    public String addPatient(HttpServletRequest request,
                               HttpServletResponse response, Model model) {
        PipelineInputCache pi = new PipelineInputCache();
        // supply http body as the source for the resource Create pipeline
        pi.setInputPort("source", request.getInputStream());
        // supply current resource URL as the base URL to craft hyperlinks
        String baseUrl = request.getRequestURL().toString();
        pi.addParameter("stylesheetParameters", new QName("baseURL"),
                        request.getRequestURL().toString());
        PipelineOutput output = m_addPatientProcessing.executeOn(pi);
        ...
    }
}
```

Runtime XProc pipeline parameter bindings
Binding Operations to XProc Pipelines

The Spring config:

```xml
<bean id="Patients" class="com.emc.cto.healthcare.Patients">
    <property name="getPatients" ref="getPatientsXMLProcessingContext" />
    <property name="addPatient" ref="addPatientXMLProcessingContext" />
    <property name="getPatient" ref="getPatientXMLProcessingContext" />
    <property name="replacePatient" ref="replacePatientXMLProcessingContext" />
    <property name="deletePatient" ref="deletePatientXMLProcessingContext" />
</bean>

<bean id="addPatientXMLProcessingContext" class="com.emc.cto.xproc.XProcXMLProcessingContext">
    <property name="xprocPool" ref="xprocPool" />
    <property name="pipelineSource"><value>classpath:resourceCreate.xpl</value></property>
    <property name="inputs">
        <map>
            <entry key="xqueryscript" value="classpath:addPatient.xq" />
            <entry key="stylesheet" value="classpath:hyperlinksPatient.xslt" />
        </map>
    </property>
    <property name="options">
        <map>
            <entry key-ref="iDAssignmentXPath" value="pat:Patient/pat:pid" />
        </map>
    </property>
    <property name="parameters"><map/></property>
</bean>
```

One XML processing context per operation

Set design time parameters into the pipeline

Configure the operation with the XProc pipeline
A Proposed Alternative
What if the developer could just write X-things?

<webService>
  <resource uriTemplate="/patients" operation="GET">
    <xProcImplementation source="classpath:resourceGet.xpl">
      <parameters>
        <xqueryScript type="file">classpath:getPatients.xq</xqueryScript>
        <hyperlinksSS type="file">classpath:hyperlinksForPatients.xslt</hyperlinksSS>
      </parameters>
    </xProcImplementation>
  </operation>
  <operation uriTemplate="/patients/{patientId}" operation="GET">
    <xProcImplementation source="classpath:resourceGet.xpl">
      <parameters>
        <xqueryScript type="file">classpath:getPatient.xq</xqueryScript>
        <hyperlinksSS type="file">classpath:hyperlinksForPatients.xslt</hyperlinksSS>
        <xqueryParameter type="uriTemplateParameter" variable="patientId"/>
      </parameters>
    </xProcImplementation>
  </operation>
  ...
</resource>
  ...
</webService>
Epilogue

• Our work on the XML REST Framework is freely available:
  – Version 1, basic framework supporting xDB and XQuery: https://community.emc.com/docs/DOC-6434
  – Version 2, adds hyperlink insertion and XSLT application: https://community.emc.com/docs/DOC-6485
  – Version 3, moves to Spring MVC and includes XProc: https://community.emc.com/docs/DOC-10494
  – Includes:
    • Framework source code
    • Sample application
    • Documentation

• XML Technologies such as XProc and XSLT are suitable mashup building tools
  • (more to come on this topic later in the program)
THANK YOU