

XML Technologies for RESTful Services Development

Second International Workshop on
RESTful Design

March 2011

Cornelia Davis

Senior Technologist

Office of the CTO

EMC Corporation

Motivation

Drivers for this work at EMC

- Goal: Establish an integration architecture that will help EMC bring together its own and partner products.



- We're basing it on REST because:
 - In order to get some standardization the principles need to be easy
 - The environment is by definition highly distributed and potentially very large and increasingly in the cloud
 - Product groups are all building RESTful interfaces
 - Resource focus
 - With media-type handling
 - With hyperlinking
 - With caching

Agenda

- REST Principles
- Implementing RESTful services
- What's next?

REST Principles

- REST is an architectural style that depends upon:
- Identification and addressibility 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



Agenda

- REST Principles
- Implementing RESTful services
- What's next?

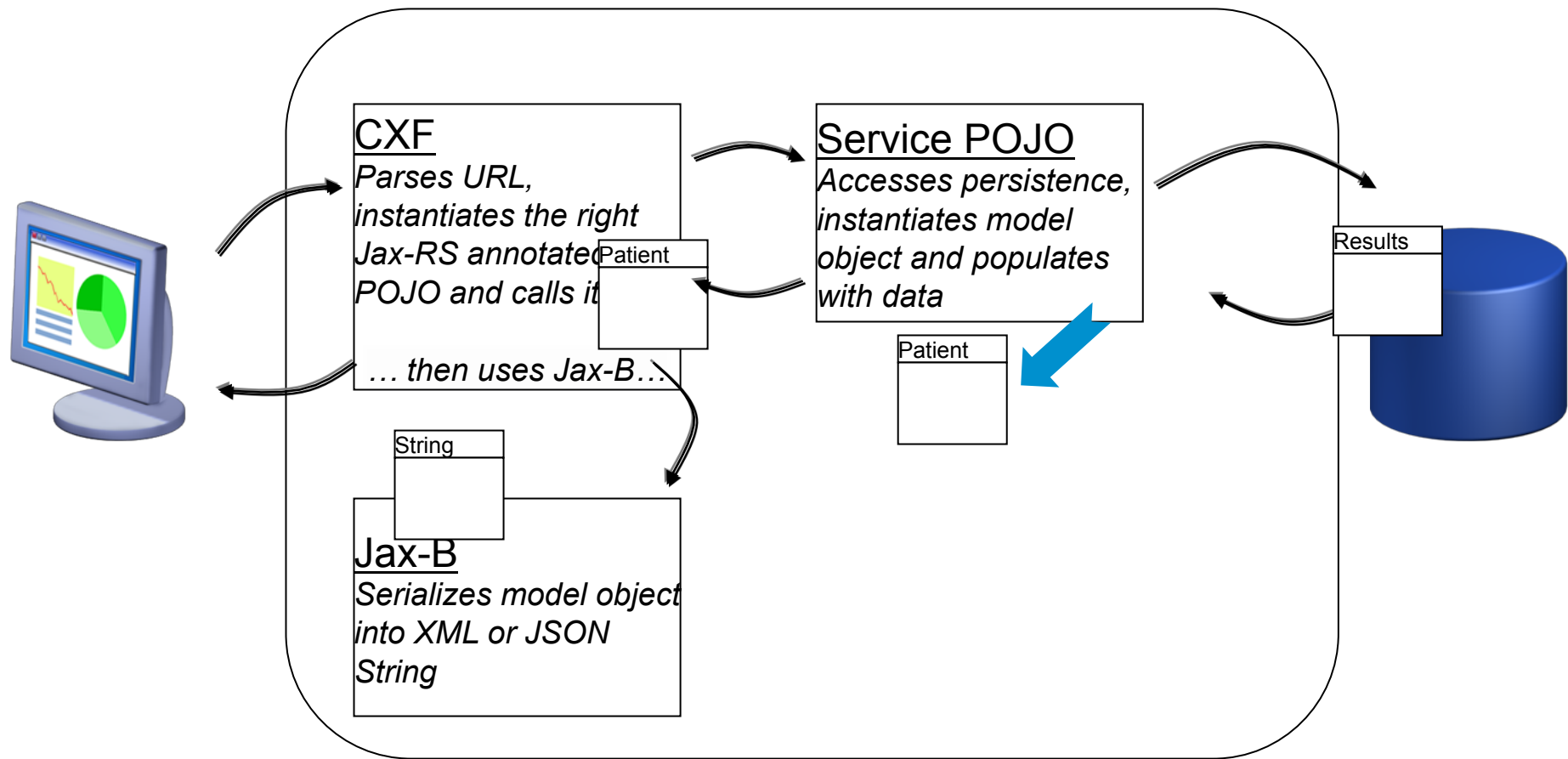
Implementing Resource using Jax-RX

- Java API for RESTful Web Services¹
- Annotations for naming, uniform interface and media types
- Several Implementations include Apache CXF², and Jersey³
- Media type support through integration with Jax-B⁴ Implementations

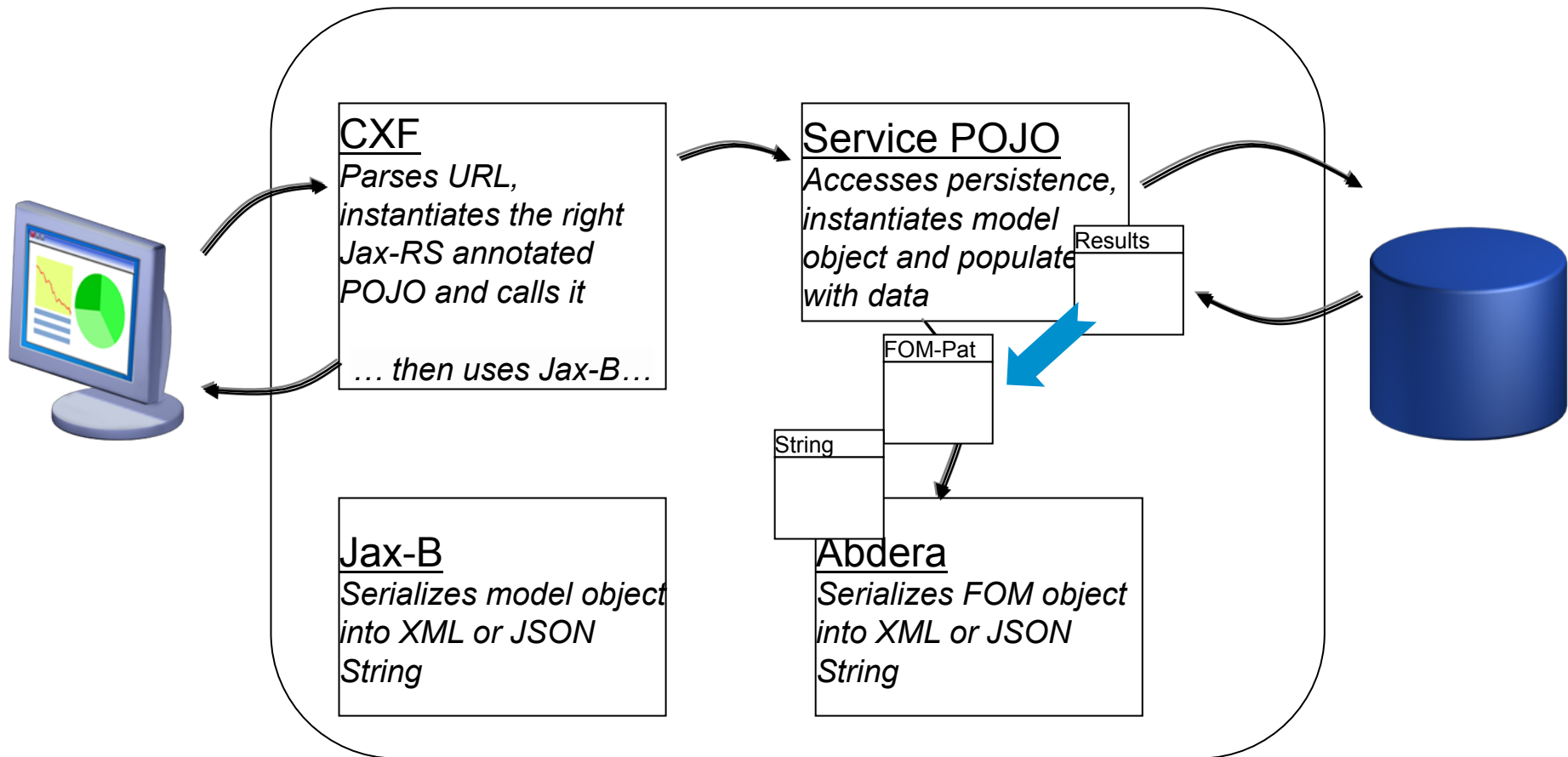
```
@Path("/patients")
public class Patients extends RestObject {
    ...
    @GET
    @Produces("application/xml")
    public Response getPatients() {
        ...
    }
    @POST
    @Consumes("application/xml")
    public Response addPatient(...) {
        ...
    }
    @Path("/{pid}")
    public Patient getPatientByID(@PathParam
    ("pid")String pid) {
        Patient patient = new Patient(pid);
        patient.setBaseURI(getSelfURI());
        return patient;
    }
}
```

¹ <https://jsr311.dev.java.net/> ² <http://cxf.apache.org/> ³ <https://jersey.dev.java.net/> ⁴ <http://jcp.org/jsr/detail/222.jsp>

RESTful Service without XML Toolset



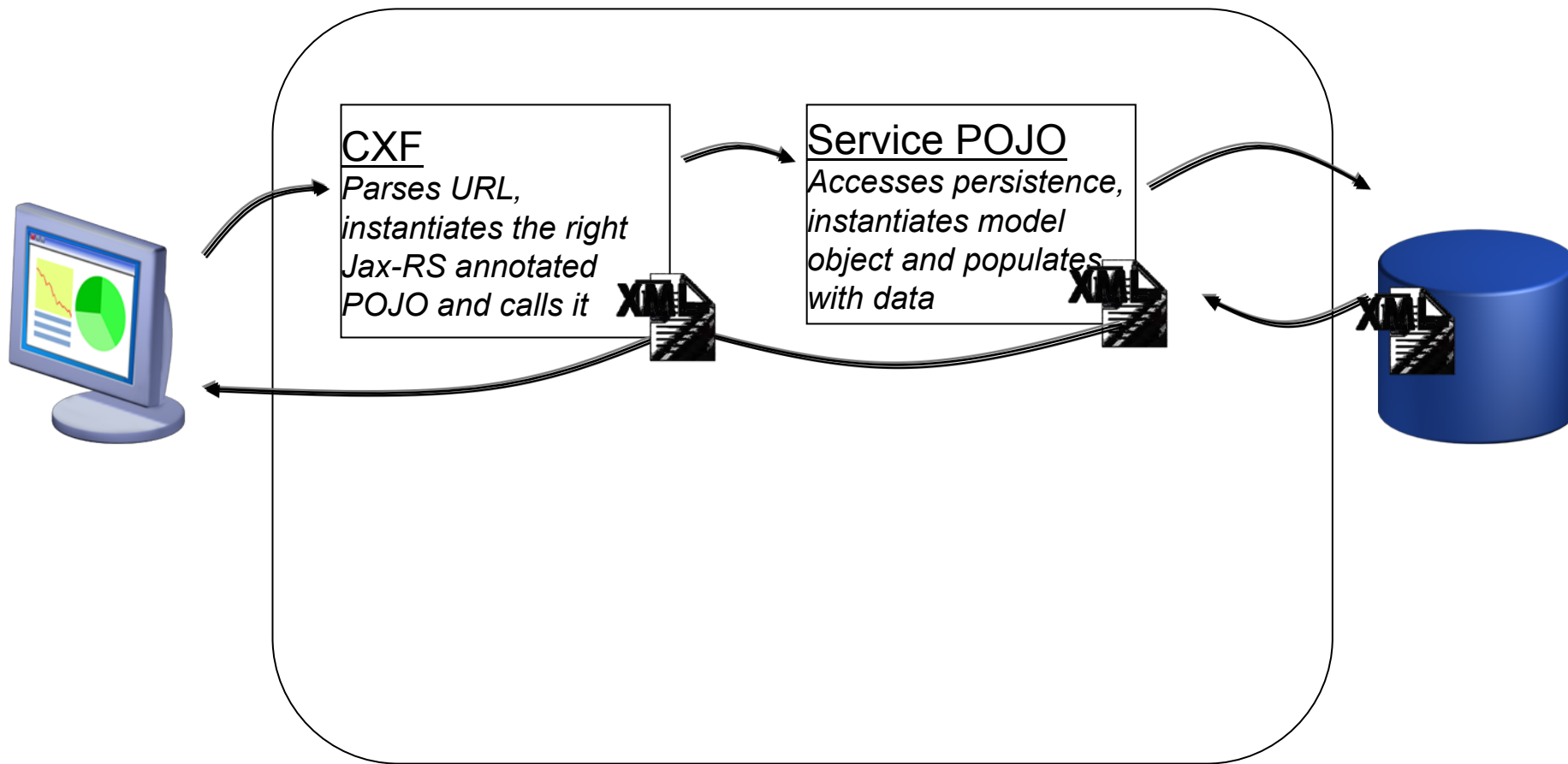
...now with Atom representations



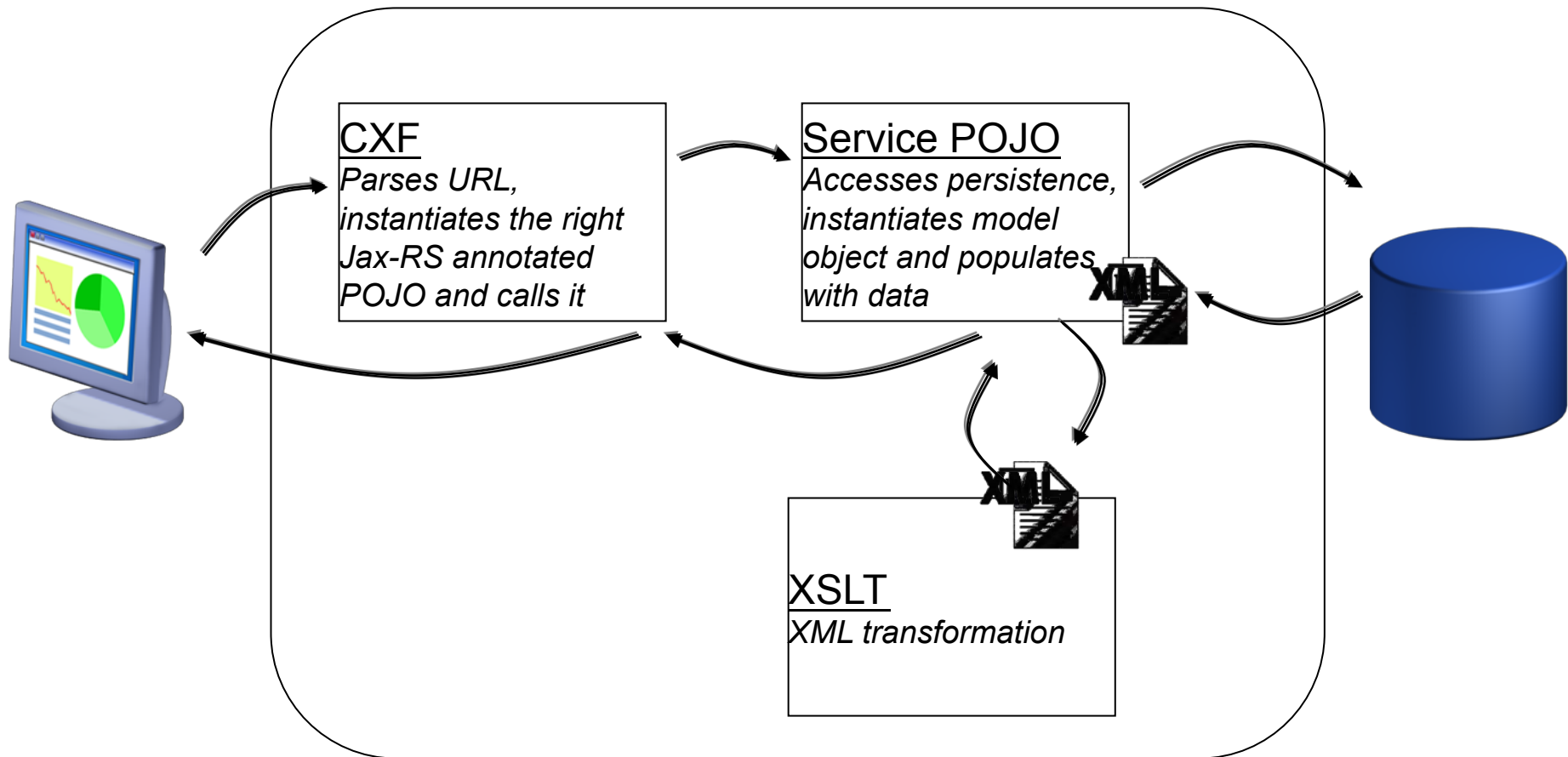
How Well Does Jax-RS Address the Core REST Principles?

Feature	Jax-RS	Comments
Named Resources	●	URI template support - <code>@Path(/Patients/{pid})</code>
Define Uniform Interface	●	<code>@GET</code> , <code>@PUT</code> , <code>@POST</code> , <code>@DELETE</code> , <code>@PATCH</code> , etc.
Handle media types	◐	<code>@Produces</code> and <code>@Consumes</code> but handles only media type (format) and not content type (schema)
Hyperlinking	○	nothing
Implementation		Generally happens with a bunch of java code.

RESTful Service – XML as the Dial Tone



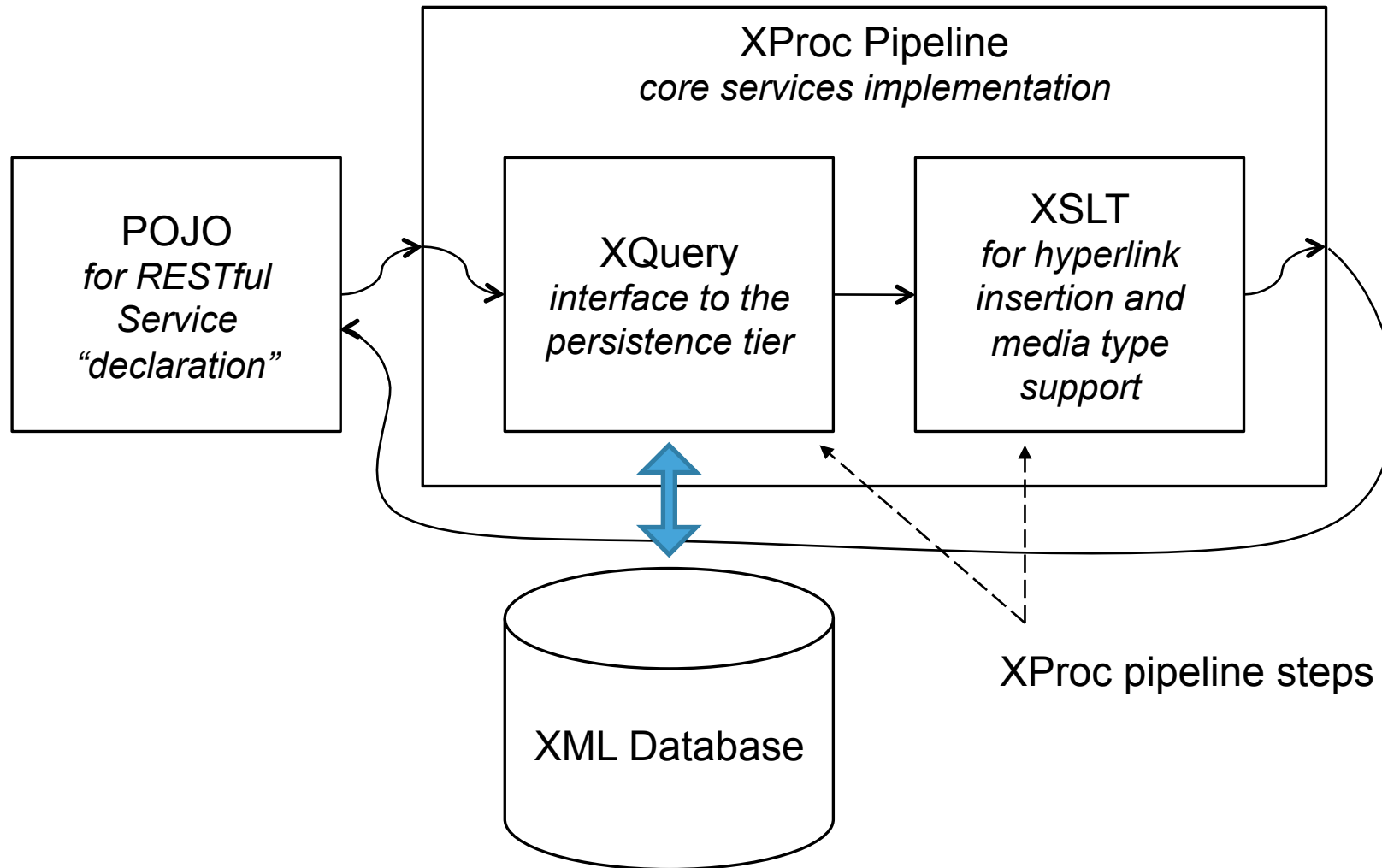
Insert Hyperlinks via XSLT Transformation



The REST XML Framework

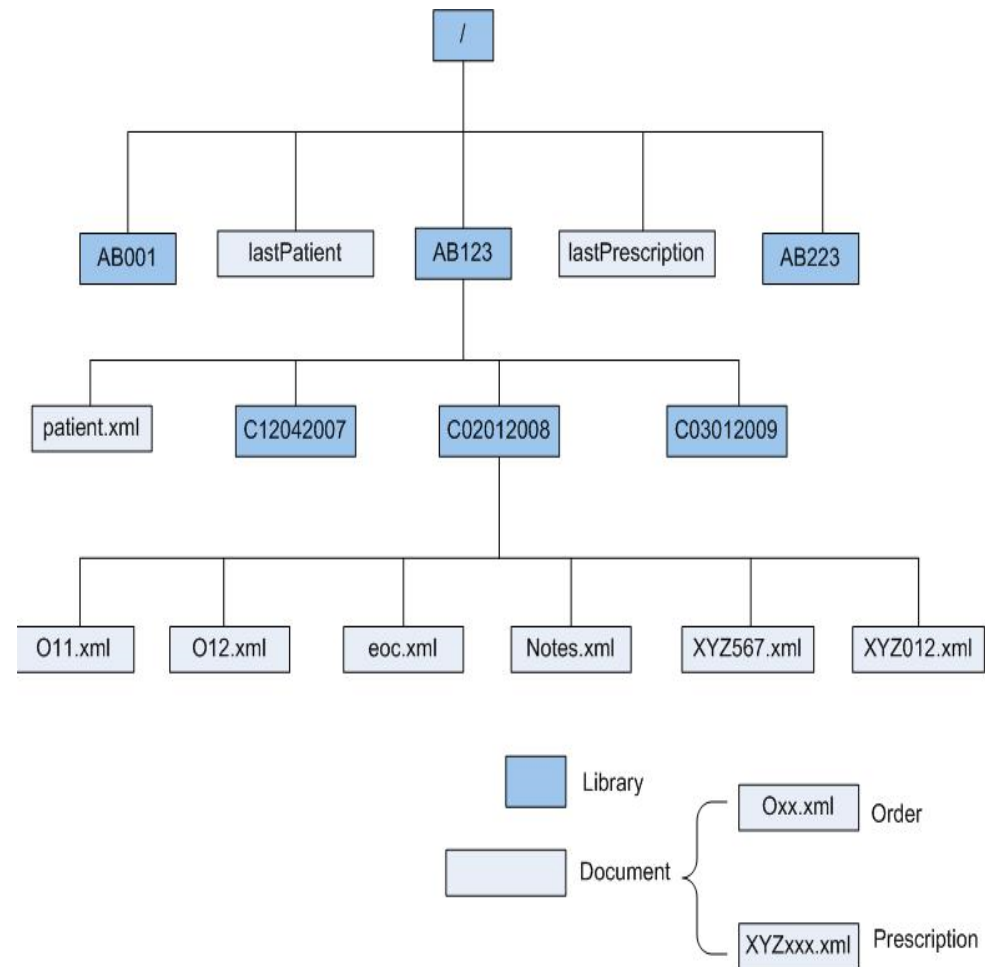
- We've developed a framework that:
 - Supports rapid development of **domain specific** RESTful Web Services
 - Addresses all four REST principles – resources, uniform interface, media types, hyperlinking
 - Promotes reusability and deployment time configuration
 - Heavily leverages XML
- Our framework embraces and extends:
 - CXF (Jax-RS implementation) or Spring MVC
 - XML Technology
 - XML Database – EMC Documentum xDB
 - XQuery Engine
 - XProc Engine - Calumet
 - XSLT
 - The Spring Framework
- Follows an XRX style
 - **X**forms on the client
 - **R**ESTful interfaces
 - **X**Query on the server
 - O'Reilly published piece: XRX: Simple, Elegant, Disruptive
http://www.oreilynet.com/xml/blog/2008/05/xrx_a_simple_elegant_disruptiv_1.html

Overview



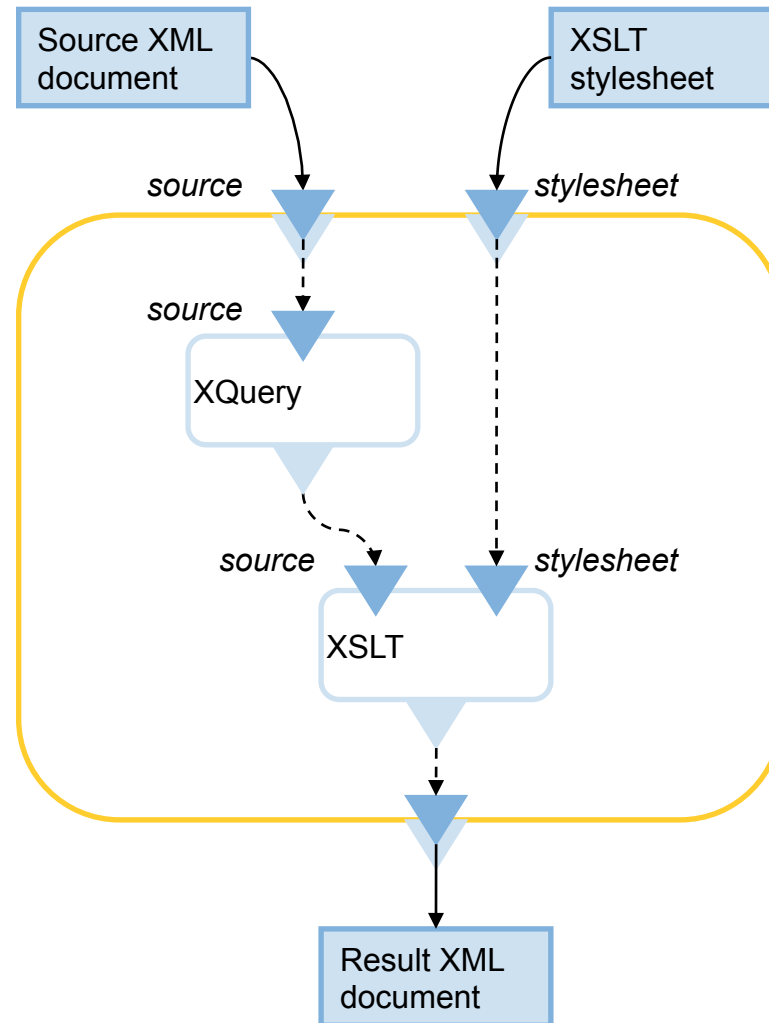
Our Persistence – EMC Documentum xDB

- Native XML database
- Designed for content-oriented applications
- Ideal for warehousing content in an application-neutral format
- More flexible than RDBMS for unstructured content and for aggregating diverse data sets
- Extensive, highly productive application development
- XQuery: powerful search, retrieval, linking, and updates
- Transaction and full ACID support
- High-performance, scalable architecture, with minimal overhead



XProc pipeline

- Steps (can) have
 - Input ports
 - Output ports
 - Options
- Parameter input ports
- Atomic vs. compound steps
- Standard atomic step library
 - xslt, xquery, http-request, xinclude, ...
- Custom steps
 - Custom compound steps
 - Step libraries



Declaring Services

- In the POJO: Encapsulate operation pipeline and design time bindings

```
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 binding

Binding Operations to XProc Pipelines

One XML processing context per operation

- The Spring config:

```
<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>
```

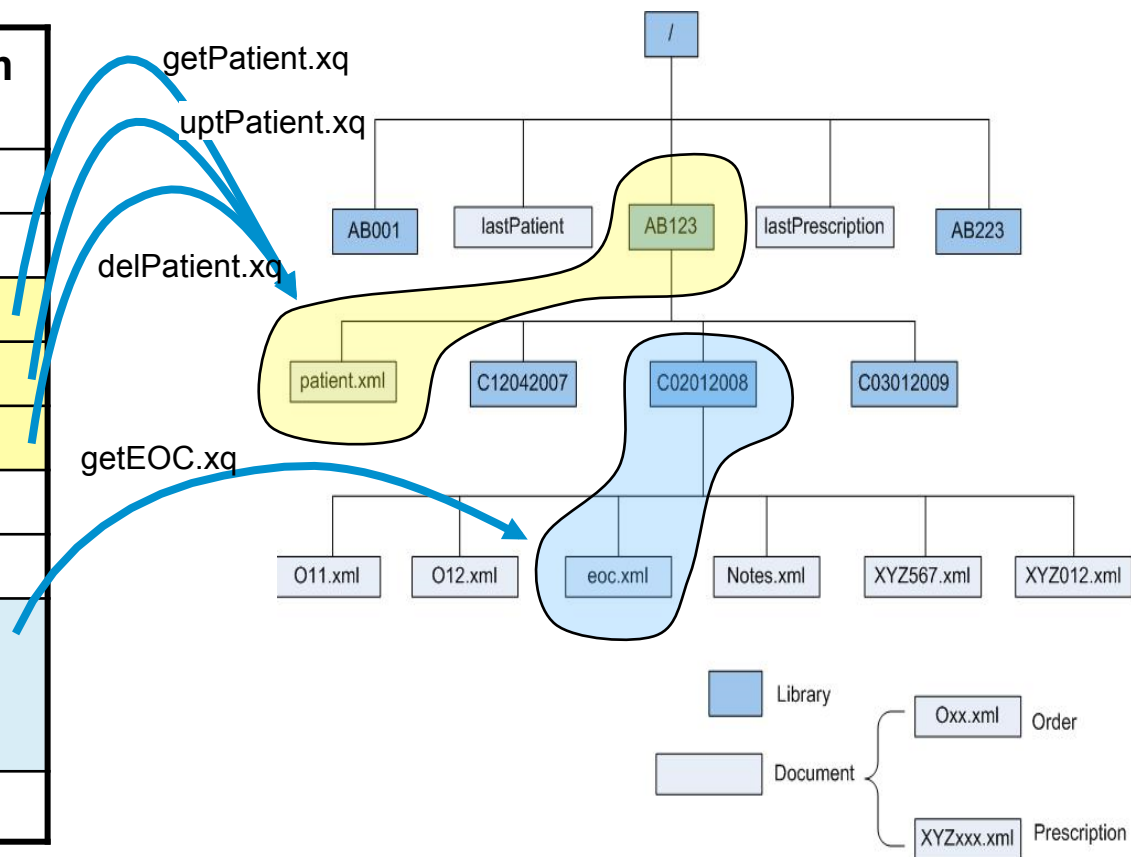
Set design time parameters into the pipeline

Configure the operation with the XProc pipeline

Binding Resource Model to Physical Model

(typically) One xQuery per operation per resource!!

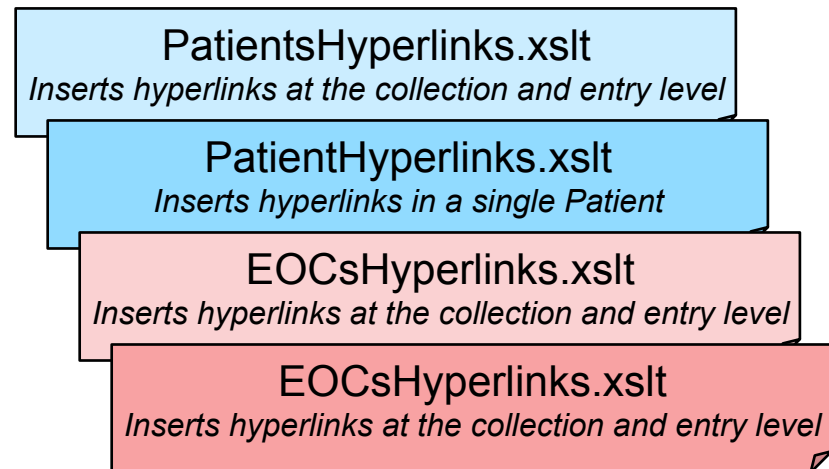
Resource	URI Template	Uniform IF
Patients	/patients	GET POST
Patient	/patients/{id}	GET PATCH DELETE
Care Episodes	/patients/{id}/careepisodes	GET POST
Care Episode	/patients/{id}/careepisodes/{cid}	GET
...		



Associating XSLT Transformations

Resource	URI Template	Uniform IF
Patients	/patients	GET
		POST
Patient	/patients/{id}	GET
		PATCH
		DELETE
Care Episodes	/patients/{id}/careepisodes	GET
		POST
Care Episode	/patients/{id}/careepisodes/{cid}	GET
...		

One XSLT per representation



...

Injected into beans

(in spring config)

...

Sample XSLT for Inserting Hyperlinks

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ... a bunch of namesp
  <xsl:import href="classpath:insertHyperlinks.xslt" />
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

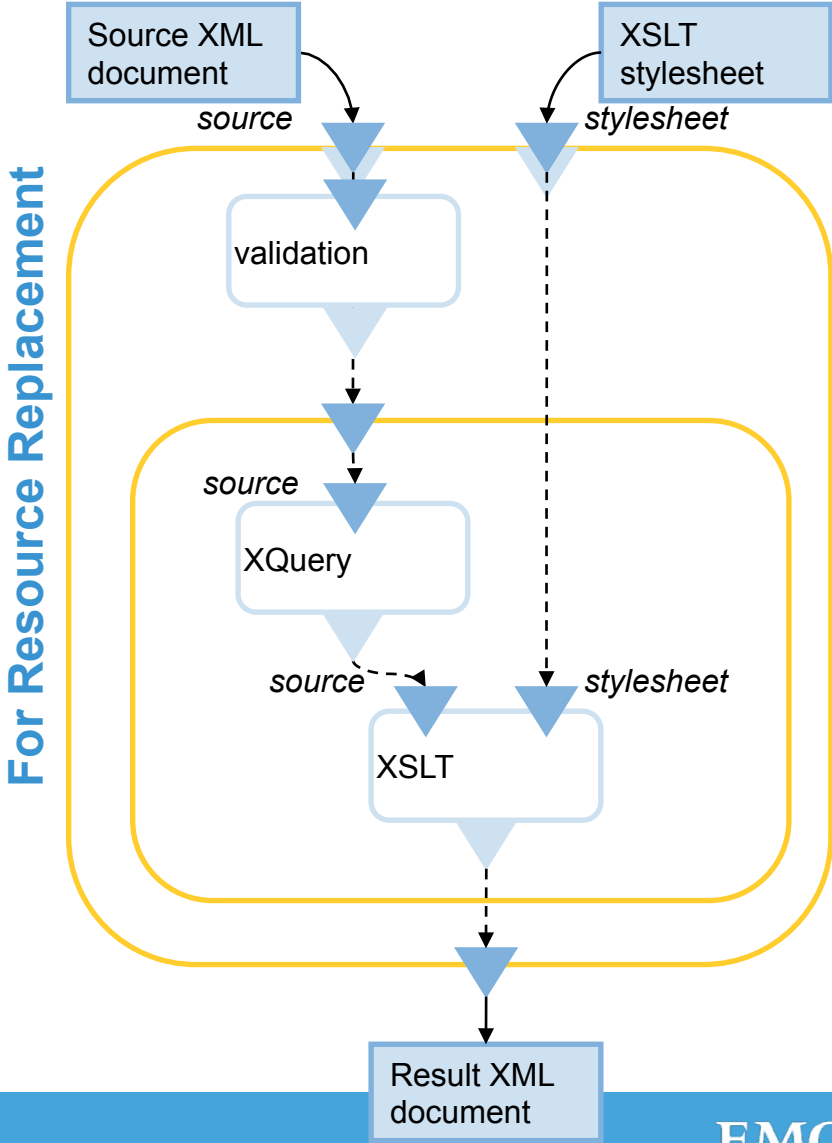
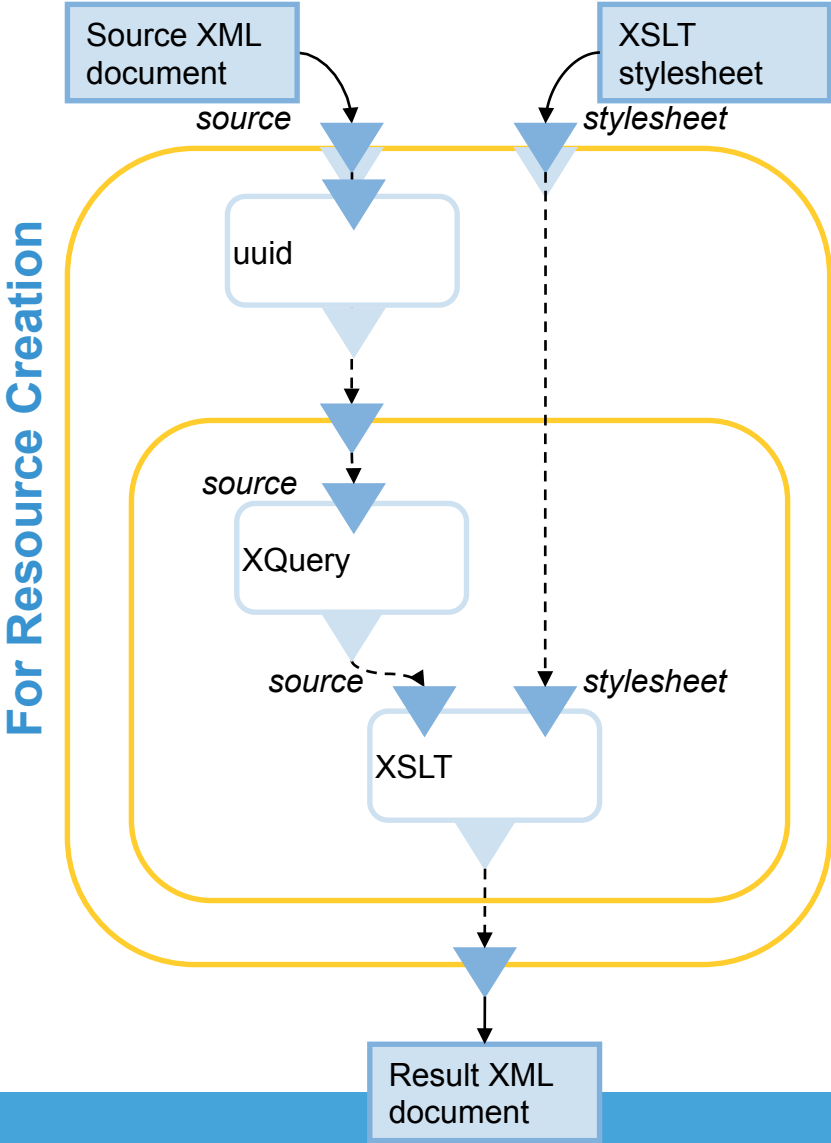
  <xsl:param name="baseURL"/>

  <xsl:template match="p:Patient" mode="insertthere">
    <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, '/')"/>
    </atom:link>
  </xsl:template>
</xslt:stylesheet>
```

XProc Pipeline Parameters Passed to Steps

```
<!-- insert hyperlinks -->
<p:xslt name="xslt">
  <p:input port='source'>
    <p:pipe step='xquery' port='result' />
  </p:input>
  <p:input port='stylesheet'>
    <p:pipe step='main' port='stylesheet' />
  </p:input>
  <p:input port='parameters'>
    <p:pipe step='main' port='stylesheetParameters' />
  </p:input>
  <p:with-param port='parameters' name='baseUrl' select='/location/text()'>
    <p:pipe step='locXML' port='result' />
  </p:with-param>
</p:xslt>
```

Template Pipelines



How Our Framework Measures Up

Feature	Jax-RS	Our FW	Comments
Name Resources	●	●	Use Jax-RS annotated POJOs
Define Uniform Interface	●	●	Use Jax-RS annotated POJOs
Handle media types	◐	◐	Currently exploring this . – Spring MVC – Stronger typing
Hyperlinking	○	●	Via XSLT
Implementation		●	XML-centric, rapid application development

Why XML-Centric?

- XML is often the format for resource representations
- XML Databases suitable for storing many data sets
 - Ragged, schema-less, evolving
- XML is a good model language
- XSLT is ideal for inserting hyperlinks
 - `<xsl:template>` defines insertion points
 - XPath for expressing hyperlink construction rules.
- XProc is a nice high-level language
 - Dealing with composite resources
 - May evolve to make “programming” accessible to the non-programmer

Agenda

- REST Principles
- Implementing RESTful services
- What's next?

Continued Areas of Exploration...

- Finite State Machine for modeling application state
- Media types
 - “You b*\$t@^d, you’ve used ‘application/xml’”
– Ian Robinson
 - Media types are part of the contract for your services
- Feed paging and archiving – RFC 5005¹
- Caching support – i.e. eTags
- JSON

¹ <http://www.ietf.org/rfc/rfc5005.txt>

THANK YOU