Apache CXF Web Services

Dennis M. Sosnoski
Portland Java Users Group
August 16, 2011

About me

Java, web services, and SOA expert

- Consultant and mentor on SOA and web services for companies worldwide
- Training instructor for Java web services (Apache Axis2 and CXF, Metro), web services security, etc.
- Open source developer:
  - Apache Apache Axis2, CXF, and WSS4J committer
  - JiBX XML data binding lead developer
- Writer on Java, XML, and SOA for IBM developerWorks
- Presenter at users groups and conferences worldwide

Aotearoa / New Zealand and U.S. based

Outline

CXF and web services background

REST web services in CXF

SOAP web services in CXF

- SOAP basics with JAX-WS
- WSDL service definitions for SOAP
- WS-* SOAP extensions

Building SOA on CXF and Apache

Support for CXF and Apache SOA

CXF web services

CXF the leading open source web services stack for Java

- Best feature support
- Most flexible approach
  - Configure using annotations
  - Manipulate configuration directly in code
  - Configure using Spring
  - Supports range of data binding tools, front ends, etc.
- Very active development team, fast bug fixes and enhancements

Quick look at CXF components...

Bus

Registry of extensions, interceptors, and properties

Provider of shared resources:

- WSDL managers
- HTTP destination factory (Jetty the default)
- Features usable by applications
- etc.

Configurable – default implementation uses Spring

```
<cxf:bus xmlns:cxf="http://cxf.apache.org/core">
    <cxf:features>
        <cxf:logging/>
        <wsa:addressing xmlns:wsa="http://cxf.apache.org/ws/addressing"/>
    </cxf:features>
</cxf:bus>
```

Front-ends

Programming model for application interactions with CXF

Three main variations:

- JAX-RS (configured with annotations and/or XML)
- JAX-WS (configured with annotations and optionally XML)
- Simple (limited compared to JAX-WS, but no annotations required)

Interceptors

Used as message processing steps

Separate lists of interceptors for different flows:

- Normal inbound
- Normal outbound
- Fault inbound
- Fault outbound

Different phases of processing used to order invocations

– **Message may be transformed in process**

Can be used for any desired special processing (e.g., Logging)

Data Bindings

Convert between XML and Java object representations

Choices allow flexibility:

- JAXB 2.x only approach supported by JAX-WS standard
- XMLBeans for flexible access to data as XML
  - DOM model allows XPath and XQuery access, other tools
  - Data binding facade for limited conversions to/from Java objects
- JiBX for flexibility
  - Bindings that handle structural differences between XML and objects
  - User extensions for handling special cases
  - Multiple bindings to same Java objects, input-only and output-only

Web services approaches

Two main schools of thought:

- REST focuses on simplicity and flexibility
- SOAP focuses on extensibility and feature support

Both approaches have their use cases

Outline

CXF and web services background

REST web services in CXF

SOAP web services in CXF

- SOAP basics with JAX-WS
- WSDL service definitions for SOAP
- WS-* SOAP extensions

Building SOA on CXF and Apache

Support for CXF and Apache SOA

REST basics

Representational State Transfer

Based on Roy Fielding's doctoral thesis on HTTP:

- Formalization of the web as resources
- HTTP verbs provide actions on resources:
  - **GET** to retrieve current state of a resource
  - **PUT** to replace the current state of a resource
  - **POST** to create a new resource
  - **DELETE** to remove a resource
- Powerful and flexible structure for resource-oriented system
  - GET verb guaranteed safe, responses can be cached
  - PUT and DELETE verbs are idempotent

Library example

Base URI http://localhost:8080/library

- http://localhost:8080/library/books to access the book collection directly
  - Using this URI operates on all books in collection
    - GET returns all books
    - PUT replaces all books
    - POST adds a new book
    - DELETE removes all books
  - http://localhost:8080/library/books/{*isbn*} operates on a particular book (GET, PUT, DELETE)

Library example continued

- http://localhost:8080/library/types to access the books by type
  - Using this URI operates on all books of type
    - GET returns all types
    - PUT replaces all types
    - POST adds a new type
    - DELETE removes all types
  - http://localhost:8080/library/types/{*name*} operates on a particular type (GET, PUT, DELETE)
- Provides flexible access to the book collection as a structured resource

JAX-RS

JAX-RS uses Java annotations for REST support

```
@Path("library")
public class RestLibraryImpl
{
    @GET @Path("books")
    public BookList getAllBooks() { ... }

    @PUT @Path("books")
    public void putAllBooks(BookList books) { ... }

    @POST @Path("books")
    public String addBook(Book book) { ... }

    @DELETE @Path("books")
    public void deleteAllBooks() { ... }

    @GET @Path("books/{isbn}")
    public Book getBook(@PathParam("isbn") String isbn) { ... }
    ...
}
```

REST client support

Current JAX-RS version does not define client handling

CXF implements its own REST client support

- WebClient interface to service uses HTTP directly
  - WebClient.create(target) to get an instance
  - “Fluent” API to modify state
    - reset() clears modifications
    - path(“...”) appends to path
    - accept(“...”) sets content accept type
    - get() / put() / post() / delete() execute operations
    - Many other variations...
- Proxy-based interface to service with JAXRSClientFactory hides details (more on this later)

JAX-RS example

REST Library service using CXF JAX-RS

- Demonstration of service in browser
- Service code walkthrough and discussion
- Client code walkthrough and demonstration
- Fill in remaining operations for client

Parameter types

Many ways to get information from request:

- `@PathParam` – portion of path mapped to parameter
- `@QueryParam` – query parameter values
- `@FormParam` – value from form data
- `@HeaderParam` – HTTP header value
- `@CookieParam` – HTTP cookie value
- `@MatrixParam` – named qualifier parameter value
  - `/library/books;author=Cook,Glen`

CXF WebClient has matching methods (query(...), form(...), header(...), cookie(...), matrix(...)

Data handling

Can specify media types for request and response bodies

- `@Consumes` for request data type
- `@Produces` for response data type
- Most interesting choices for web services:
  - `"application/xml"` for standard XML
  - `"application/json"` for JSON

Providers used for serializing / deserializing Java data

- `@Provider` annotation
- Can also be done using Spring or in-code configuration

CXF allows all data bindings to be used with JAX-RS

User models

CXF lets you supply XML service description in place of JAX-RS annotations:

- <jaxrs:model> root element as wrapper
- <jaxrs:resource> child element for each resource class
  - path, produces, consumes, verb attributes
  - <jaxrs:operation> child element for each resource method
    - path, produces, consumes, verb attributes
    - Optional <jaxrs:param> child element for each parameter

Allows plain Java class to be exposed as service

Debugging and tracking problems

Monitor message exchanges externally

- Tcpmon simple, fast, and easy tool for web services work
- Wireshark excellent for all types of transport protocols, but not as simple
- soapUI has advanced features, including automated testing

Logging to track internal processing flow

- Can add interceptors to view messages being exchanged
- Output controlled by logging configuration file

Debugging through code (both client and server side)

- Can run services within IDE
- IDE can attach to server via JPDA

Demonstration using proxy-based client

JSON formatting

JSON format can be an issue

- Differences over array handling (name around items?)
- Differences over object handling (name around values?)

CXF gives you options:

- Transformation feature provides scripted control
  - Basic transformations of both XML and JSON
  - Configured via Spring or in code
- Can also use Jackson JSON handling
  - Configure as provider for CXF JAX-RS

WADL service descriptions

Web Application Description Language

- <wadl:application> root element with optional child elements
  - <wadl:grammars> child defines document grammars
  - <wadl:resources>child defines resources with nested structure
    - <wadl:resource> gives path (if any)
      - <wadl:param> children for parameters
      - <wadl:method> children for methods
      - <wadl:resource> children for subresources
  - <wadl:method> gives request/response information

View example for Library

Proposal to W3C, no plans to make it an official standard

CXF supports client code generation

REST benefits

Simple interface for working with resources:

- Clients easily implemented directly in code
- Browser-compatible, to some degree
- JSON supports easy handling in AJAX clients
- Resources may be cached for GET responses
- GET, PUT, DELETE just retry on failure
- HTTPS/TLS can be used for point-to-point security

REST limitations

Not all that's called REST really is REST:

- Random query parameters slapped on a URI
- Side-effects from GET operations
- POST operations to modify existing data

Even when used correctly, difficult to handle services:

- Consider all the different resources involved in making an airline reservation
- No direct way to map this to a REST architecture

Outline

CXF and web services background

REST web services in CXF

SOAP web services in CXF

- SOAP basics with JAX-WS
- WSDL service definitions for SOAP
- WS-* SOAP extensions

Building SOA on CXF and Apache

Support for CXF and Apache SOA

SOAP basics

Formalized message exchanges at the core

SOAP defines standard wrapper for all messages

- Envelope is wrapper for content, but no useful information
- Header can contain both application data and control information
- Body contains application data as XML
- Each request message identifies the operation to be performed by service

Normally a single URI for service, only POST verb used

Biggest advantage is extensibility

- Designed with extension in mind
- Supports plug-in extension technologies

The WSDL additive

Web Services Description Language

- Defines service interface as XML exchanges
- Extensible for other types of metadata

Clearly defines the service interface

- Different access techniques (transport, technology)
- Operations defined by each port type
- Input / output messages for each operation
- Detailed XML description of each message

Supports “automatic” configuration of clients and providers

Contract adherence can be verified by tools

Library example

Service URI http://localhost:8080/library

- getAllBooks operation to return all books in collection
- getBook operation to get a single book
- addBook operation to add a book
- Any other convenient operations

No formal structure to operations defined by service

WSDL service definition walk-through

JAX-WS

Standardized SOAP services in Java

- Uses source code annotations
  - Define interface representing service
  - Attach actual implementation to interface
  - Supply pregenerated WSDL for the service
  - Many additional options
- Reference implementation uses JAXB data binding

Several open source implementations:

- Sun/Oracle reference implementation supplied with Metro
- CXF supports with all data bindings, attachments, etc.

JAX-WS example

JAX-WS uses Java annotations for SOAP support

```
@javax.jws.WebService
  (endpointInterface="com.talend.ws.library.soap.common.Library",
  portName="library",
  targetNamespace="http://ws.talend.com/library/wsdl",
  serviceName="Library")
public class LibrarySoapImpl implements Library
{
    public boolean addBook(String type, String isbn,
        List<String> author, String title) { … }

    public BookInformation getBook(String isbn) {
        return m_server.getBook(isbn);
    }

    public List<BookInformation> getBooksByType(String type) {
        return m_server.getBooksByType(type);
    }
}
```

JAX-WS usage

Most often used with existing service definition

- Generate JAX-WS service and/or client code (including annotations) from service definition
- Add configuration information for the stack (CXF)
- Deploy using the stack

Some peculiarities

- Client code generally requires access to WSDL at runtime
- Runtime WSDL processing slows startup

Code generation from WSDL

CXF provides Wsdl2Java tool

- Run directly, from Ant, or via Maven plugin
- Many options, including:

  -validate – validate the supplied WSDL before generating switch

  -p package – target package for generated code

  -client – only generate client code switch

  -server – only generate server code switch

  -wsdlLocation path – path used for WSDL in generated code

  -d dir – output directory

Example generates both client and server code at once

JAX-WS example

SOAP Library service using CXF JAX-WS

- Demonstration of client and service
- Monitor message exchange with Tcpmon
- Client and service code walkthrough and discussion
- Implement added operations for service
- Deploy and test to confirm

SOAP Faults

Fault element part of the basic SOAP definition

- Replaces normal Body content for response
- Way to signal processing errors

Basic Fault structure uses predefined error codes

Also allows arbitrary content in <detail> element

Application-level Faults are defined in WSDL:

- Faults are listed in two (or more) places
  - Within the relevant <portType>/<operation>
  - Within the relevant <binding>/<operation>
- The name attributes must match! (no namespaces)

Other JAX-WS options

Annotation-based configuration of JAX-WS handlers

Dispatch/Provider API for working directly with XML

- Work with payload or entire message
- DOM, stream, transform Source, etc.

CXF extension supports alternative data bindings:

- JAXB 2.x the standard
- XMLBeans useful when working with data as XML
  - XPath/XQuery and DOM for XML manipulation
  - Data binding facade for working with data
- JiBX data binding useful for flexibility

Outline

CXF and web services background

REST web services in CXF

SOAP web services in CXF

- SOAP basics with JAX-WS
- WSDL service definitions for SOAP
- WS-* SOAP extensions

Building SOA on CXF and Apache

Support for CXF and Apache SOA

Where to start?

SOAP web services can be developed in different ways

- “WSDL first” approach develops WSDL first, generates code from WSDL for both client and service
- “Code first” approach normally means exposing service implementation code directly
- Another alternative is “Code to WSDL”, starting from existing code to develop WSDL

WSDL structure

Understand references in WSDL

- <service>/<port>/@binding references <binding> name
- <binding>/@type references <portType> name
- <portType>/<operation>/<input>|<output>|<fault>/@message references <message> name
- <message>/<part>/@element references <element> name in schema definition

All these references use namespaces

Example

Getting started

“WSDL first” fine, but difficult

- Complex and confusing WSDL structure
- Poor tools for editing and refactoring WSDL and schema

“Code first” approaches exposing code directly create tight couplings

- Changes in service code flow through to client
- Existing clients may be broken by changes

“Code to WSDL” often best approach

- Use existing service code as base
- Generate WSDL from code, then modify as appropriate
- Generate code from WSDL, adapt existing code to match

Code to WSDL tools

CXF provides java2wsdl tool, some limitations

- No way to customize handling
- No way to generate documentation

JiBX project Jibx2Wsdl often a better alternative

- Generates JiBX binding, schema, and WSDL from supplied service class(es)
- Extensive customizations to control generation
- Uses JavaDocs from source code for WSDL and schema documentation
- Lets you leverage your code investment for web services, even if not using JiBX

Service extensibility issues

Hard to keep services frozen

- Interface changes to suit enterprise requirements
- Data content expands

Changing the service interface is difficult

- Many changes break existing clients

Adding new versions also problematic

Outline

CXF and web services background

REST web services in CXF

SOAP web services in CXF

- SOAP basics with JAX-WS
- WSDL service definitions for SOAP
- WS-* SOAP extensions

Building SOA on CXF and Apache

Support for CXF and Apache SOA

SOAP extension technologies

Extends basic SOAP with added functionality:

- WS-Addressing – adds message identification and routing information
- WS-Security – adds XML encryption and signing
- WS-Trust, WS-SecureConversation – enterprise authentication and ongoing exchanges
- WS-ReliableMessaging – adds message acknowledgement and retransmission

Use SOAP Header for added metadata

All operate (more-or-less) transparently:

- Plug into SOAP stack message flow
- Little or no change to application code required

WS-Addressing

Standard for identifying messages and endpoints

- Allows messages to be assigned identifiers
- Allows messages to be correlated with other messages
- Defines endpoints involved in message exchange:
  - wsa:To for message destination
  - wsa:From for message source
  - wsa:ReplyTo for response message destination
  - etc.
- Embeds identification of desired operation in message

Frees SOAP from the request-response pattern of HTTP

Allows asynchronous services, along with many other capabilities

Web service security

Different applications have different needs

- Message confidentiality (secrecy)
- Access authentication
- Message integrity and authenticity

Security usage determined by your needs

REST can use secure transport (SSL/TLS)

SOAP has more flexibility

- Secure transport for point-to-point security
- WS-Security with intermediaries, digital signatures
- WS-Trust and WS-SecureConversation for authentication and efficiency
- WS-Policy and WS-SecurityPolicy to configure

Simple WS-Security example

AsymmetricBinding policy

- Client and server each have a key and a certificate
- Each uses private key to sign messages being sent (as desired), other's public key (certificate) to verify signatures
- Each uses other's public key (certificate) to encrypt messages being sent (as desired), own private key to decrypt messages received

Demonstrate with message capture

Discuss configuration and operation

WS-ReliableMessaging

Supports reliable message exchange

- Guaranteed delivery and ordering features
- Acknowledgments of messages received
  - May be piggy-backed on application messages using headers
  - May be sent separately (as for one-way services)
- All the issues of any message queuing system
  - Messages must be held by sender until acknowledged
  - Persistent storage needed for robust operation

Builds on WS-Addressing (endpoints, in particular)

CXF summary

Already the most flexible Java web services stack

- Best and most complete REST support
- SOAP WS-* support close to the best (and improving)
- Different configuration options adapt to widest range of application scenarios
- OSGi support adding even more flexibility

Also the best supported

The best approach for building web services!

Outline

CXF and web services background

REST web services in CXF

SOAP web services in CXF

- SOAP basics with JAX-WS
- WSDL service definitions for SOAP
- WS-* SOAP extensions

Building SOA on CXF and Apache

Support for CXF and Apache SOA

ActiveMQ

Industrial-strength message queuing

Java implementation with full JMS support

Extensive cross language clients and protocol support

Advanced message queue features (message groups, virtual destinations, etc.)

Pluggable transport protocols

Fast JDBC persistence

REST API, along with CXF and Axis2 support

Apache Camel

Powerful integration framework

Supports full range of enterprise integration patterns

- Routing and mediation rules to control processing
- Multiple ways to define the rules:
  - Fluent API for Java code
  - Spring Framework XML configuration
  - Scala Domain Specific Language (DSL)
- All three approaches support IDE smart completion

Supports wide range of endpoints and messaging models

- Identified by URIs for extensibility
- Modular so only those used need to be in classpath
- Core framework kept small

Camel CXF support

CXF supports Camel transport:

- Use as alternative to built-in CXF transports
- Allows wide range of special mechanisms (files, FTP, SMTP, etc.)

Camel supports CXF endpoints:

- Route incoming request messages to CXF
- Response processed directly by Camel

ServiceMix

Enterprise Services Bus based on Apache components

- Karaf OSGi server runtime
- CXF web services
- ActiveMQ message queue
- Camel routing and mediation
- ODE BPEL orchestration

Lightly glued together with some added code

Outline

CXF and web services background

REST web services in CXF

SOAP web services in CXF

- SOAP basics with JAX-WS
- WSDL service definitions for SOAP
- WS-* SOAP extensions

Building SOA on CXF and Apache

Support for CXF and Apache SOA

Support

Good basic support via Jira and mailing lists

Commercial support from many sources:

- CXF – my company for fast support, training, mentoring, new features; Talend for contracted support services
- ActiveMQ – Savoir Tech for fast support; can also provide contracted support, as can Talend, FuseSource, SpringSource, etc.
- Camel – Savoir Tech, Total Transaction Management for fast support; these or Talend, FuseSource, SpringSource for contracted support

Full service alternatives (including ServiceMix or alternative ESBs):

- FuseSource, SpringSource, Savoir, Talend

Resources

CXF project home: http://cxf.apache.org

- Extensive online-only documentation
- Support page http://cxf.apache.org/support.html

My web site: http://www.sosnoski.com / http://www.sosnoski.co.nz