

JiBX – XML Data Binding and Web Services

Dennis M. Sosnoski

Outline

- Introduction to JiBX data binding
 - Basic concepts
 - Flexibility
 - Performance
- JiBX Web services
 - JiBX data binding with Axis2
 - JibxSoap
 - Jibx2WsdI
 - WsdI2Jibx

Introduction to JiBX

- Project started in 2003
- First production release end of 2005
- JiBX binding definitions specify conversions
 - You define how XML relates to Java classes
 - Binding definition is itself an XML document
 - Uses tree structure for both Java classes and XML
 - Root object to root element correspondence required
 - Below root level, elements and objects are independent
- **A sample JiBX binding definition**

<mapping> elements

- <mapping> elements key to JiBX bindings
 - Concrete mapping ties XML element to Java class
 - Must be a one-to-one relationship (at least in context)
 - Abstract mapping a reusable type definition
 - Can be used directly, without an element name
 - Element name can be specified at point of use

```
<mapping name="conference" class="com.sosnoski.jibxdemo.data.Conference">
  ...
</mapping>
<mapping name="speaker" class="com.sosnoski.jibxdemo.data.Speaker">
  ...
</mapping>
<mapping abstract="true" class="com.sosnoski.jibxdemo.data.Seminar">
  ...
</mapping>
```

<structure> elements

- Structure elements the bones of the binding
 - With name supplied, adds element to XML
 - With field or get/set supplied, adds object to data structure
- With children, acts as grouping
- Without children, can reference <mapping>

```
<mapping name="speaker" class="com.sosnoski.jibxdemo.data.Speaker">
  ...
  <structure name="name">
    ...
  </structure>
  ...
</mapping>
```

<value> elements

- Simple text value representation
- Can be element content, attribute value, or inline text (i.e., between two elements)

```
<mapping abstract="true" class="com.sosnoski.jibxdemo.data.Seminar">
  <value style="attribute" name="number" field="m_number"/>
  <value style="attribute" name="speaker" field="m_speaker" ident="ref"/>
  <value name="topic" field="m_topic"/>
  <value name="date" field="m_date"/>
  <value name="time" field="m_time"/>
  <value name="length" field="m_length"/>
</mapping>
```

<collection> elements

- Repeated values
 - Correspond to Java array, collection, or equivalent
 - Items can be of specified type, inferred type, or any object type
 - Array items can be primitive types

```
<mapping name="conference" class="com.sosnoski.jibxdemo.data.Conference">
  <collection field="m_speakers"
    item-type="com.sosnoski.jibxdemo.data.Speaker"/>
  ...
</mapping>
<mapping name="speaker" class="com.sosnoski.jibxdemo.data.Speaker">
  ...
  <collection field="m_specialties">
    <value name="specialty" type="java.lang.String"/>
  </collection>
</mapping>
```

Sample application

- Conference document with multiple seminars:
 - Read in conference document
 - Count number of seminars for each speaker
 - Check for time conflicts between seminars for each speaker
 - Sort seminars by number
 - Write back out in reordered form

Demonstration

- Binding compiler usage
- Runtime usage
- Modifying bindings to change XML
- Extending handling with user code

Schema support

- W3C XML Schema (unfortunately) a standard
 - Support required for many applications
 - Hideous complexity of schema a problem
- JiBX schema support limited
 - Basic support for common data types and features
 - Limited (but soon to be improved) support for schema generation from code+binding
 - Limited (but soon to be improved) support for code (and binding) generation from schema

Derived types

- Schema allows type derivation by extension or restriction
- Derived types can substitute for base types in instance documents
 - Using `xsi:type` metadata in document to specify the actual global type name (type substitution)
 - Using substitution groups, where a global element can substitute for another element of base type
- JiBX currently supports substitution groups, not type substitution

JiBX summary

- JiBX benefits:
 - Works with existing Java data models
 - Bytecode enhancement flexibility
 - Avoids reflection runtime overhead and restrictions
 - True POJO support (no get/set methods required)
 - Multiple XML versions with same data model
 - Highest performance of any Java approach
- JiBX limitations:
 - Code+binding generation from schema weak
 - Debugging through added bytecode can be difficult

Web services

- Web services ever more important
 - Growing functionality with WS-* layers for SOAP
 - Growing use of XML data exchange even without SOAP (AJAX)
- JiBX support with three frameworks:
 - Apache Axis2
 - JibxSoap
 - XFire

Apache Axis2

- Apache web services support since 2001
- Axis2 the latest generation
 - Core is flexible framework for SOAP infrastructure
 - Supports modular framework extensions
 - Supports pluggable data binding for XML payloads
- 1.2 release April, 2007

Why Axis2?

- Axis(1) in widespread use, but limitations
 - Did not live up to performance expectations
 - Difficult to enhance and extend
 - Built on top of JAX-RPC API
 - Too many iterations of bug fixes
- Both these areas important for future
 - Web services performance a serious concern
 - Need real data binding for good schema support
 - Extensions for new crop of WS-* standards

Axis2 data binding

- Intended to allow pluggability
 - Originally shipped with XMLBeans included
 - “Axis Data Binding” (ADB) as built-in alternative
 - JiBX support implemented as of Axis2 1.0
 - JAXB 2.0 support (partial) in 1.2 release
- Implemented via extensions to WSDL2Java
 - Takes input WSDL service description
 - Generates linkage code (and data model, for most frameworks) as output

Building web services

- Start with code, or start with WSDL?
 - Start with code convenient, but can be problematic
 - Different frameworks use different XML conventions
 - Generated schemas may not reflect data semantics
 - WSDL (and schema) should be the real definition
 - But difficult to work with directly
 - Best approach depends on starting point
 - If you have schema definitions, start by building WSDL and then tune for generated code
 - If you have code, start there and tune generated WSDL

JiBX in Axis2

- WSDL2Java generates linkage code
 - Need to supply your own data classes and binding
 - Binding must define elements referenced in WSDL
 - Your code works with your own data classes
- Demonstrate code generation and options

JiBX benefits with Axis2

- Support full unwrapping of operations
- Support collection classes (not just arrays)
- Work with existing code
 - Best approach to start-from-code using Jibx2WsdI
 - Though current limitation when it comes to exceptions

Jibx2Wsd

- New JiBX-based tool (public release pending)
- Generates JiBX binding, schema, and WSDL from supplied service class(es)
- Extensive customizations to control generation
- Uses JavaDocs from source code, if available
- Best way from code to WSDL, even if not using JiBX

Basic usage

- Jibx2WsdI command line parameters:
 - b name – name for generated binding definition
 - f path – customization file path
 - p path – class loading path component
 - s path – source loading path component
 - t path – target directory for generated output
 - x class,class,... – extra classes to include in binding
 - class class ... – service classes to generate from

Jibx2WsdL customizations

- Nested customizations structure as XML
 - Defaults inherited through nesting
 - Selective overrides at each level
- `<custom>` element at root
 - Unique attributes include “add-constructors”, “direction”, etc. (`<binding>` element attributes)
 - Can have `<package>`, `<class>`, and `<wsdl>` child elements
- `<package>` element has name attribute
 - Package name, relative to containing `<package>`

<class>

- <class> element attributes include:
 - “name” – class name relative to containing <package>
 - “element-name” – for use as an XML element
 - “type-name” – for use as an XML type
 - “include”/“exclude”, “requires”/“optionals” – member name lists (space separators, member names derived from field names or property names)
- Children are <field>, <property>, <collection-field> and <collection-property> elements

Members

- `<field>` has “field” attribute (field name)
- `<property>` has “get-name” and “set-name” attributes (method names)
- `<collection-XXX>` adds “item-type” and “item-name”
- All have attributes for “element” or “attribute” (with name as value), “required”, “create-type”, and “factory”

Nesting

- All nesting elements (<custom>, <package>, <class>) also support nesting attributes:
 - “value-style” – set element or attribute style
 - “property-access” – set field or property default
 - “strip-prefixes” / “strip-suffixes” – set field name conventions
 - “require” – set “none”, “objects”, “primitives”, or “all” required
 - “name-style” – set name conversion style as “camel-case”, “hyphenated”, “dotted”, etc.

WSDL

- `<wsdl>` element has `<service>` child elements
 - “service-base” attribute for base address to access services
- `<service>` element has attributes:
 - “class” – required class name
 - “service-name” – used as base for others by default
 - “port-name”, “binding-name”, “wsdl-namespace”, etc.
 - “includes”/“excludes” – methods to be exposed

Usage

- Defaults at every level – details only needed if defaults don't work for you
- Can specify attributes of `<custom>` element and `<wsdl>` element on command line using “`--strip-prefixes=m_`” parameter, for instance
- [Demonstration of Jibx2WsdI](#)

Operations

- `<service>` has `<operation>` child elements
- `<operation>` has attributes related to method:
 - “method” – required method name
 - “operation-name”, “soap-action”, “request-message”, “request-wrapper”, “response-message”, “response-wrapper”, etc.
- Child `<parameter>`, `<collection-parameter>`, `<return>`, `<collection-return>` elements

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

Axis2 Fault handling

- Axis2 needs unique data object for each Fault
 - Data object corresponds to Fault message content
 - WSDL2Java code must create linkages
 - MessageReceiver on server catches exception to convert
 - Stub on client converts <detail> to object, creates and throws exception
 - Net effect is server exception “transferred” to client
- Ugly approach for JiBX, since blocks direct code use

JibxSoap

- JibxSoap promises speed and flexibility
 - JibxSoap uses JiBX data binding exclusively
 - JiBX data binding the fastest choice
 - Allows you to work with existing classes (or generate and modify)
- Working toward best ease of use
 - Services need only deployment descriptor
 - Client stubs with wrapped doc/lit support
- But slow progress to real release!

SOAP performance

- Comparing SOAP performance for Axis2 vs. JibxSoap
- **Demonstration**

JiBX development

- Supporting tools are outdated
 - New versions for 1.2
- Other features in development:
 - Schema type substitution support (complex types)
 - Convert JAXB 2.0 annotations to JiBX bindings?
- Starting work on JiBX 2.0
 - Source code enhancement as alternative to bytecode enhancement
 - Greatly improved schema support