Data Binding Unleashed for Composite Applications

Raymond Feng
Luciano Resende
Apache Tuscany & Nuvem committer
Agenda

• Introduction
  – Data binding
  – SCA Composite application
  – Apache Tuscany project
• Data bindings in a composite application
• Tuscany data binding framework
• Extending Tuscany data binding framework
Introduction
Understanding the concepts: Data binding and SCA composite application
What’s a data binding?

• A data binding (in this talk) denotes how business data are represented in memory as Java objects.
• For example, we can represent a customer as:
  – JavaBean (customer.Customer)
  – JAXB (customer.Customer)
  – SDO (DataObject or customer.Customer)
  – StAX XMLStreamReader
  – DOM Node (org.w3c.dom.Node)
  – XML String/byte[]/InputStream
  – JSON String/byte[]/InputStream
  – org.json.JSONObject
  – …
• The same information with different representations
SCA composite application

- SCA (Service Component Architecture, being standardized at OASIS)
  - Composite (a collection of collaborating components)
    - Component (encapsulation of reusable business logic)
      - Implementation (the code/script)
      - Service (the function it provides)
        » Interface
        » Binding (how is the service exposed)
      - Reference (the function it consumes)
        » Interface
        » Binding (how is the service accessed)
What is Apache Tuscany?

- Apache Tuscany (http://tuscany.apache.org) implements Service Component Architecture (SCA) standard. With SCA as its foundation, Tuscany offers solution developers the following advantages:
  - Provides a model for creating composite applications by defining the services in the fabric and their relationships with one another. The services can be implemented in any technology.
  - Enables service developers to create reusable services that only contain business logic. Protocols are pushed out of business logic and are handled through pluggable bindings. This lowers development cost.
  - Applications can easily adapt to infrastructure changes without recoding since protocols are handled via pluggable bindings and quality of services (transaction, security) are handled declaratively.
  - Existing applications can work with new SCA compositions. This allows for incremental growth towards a more flexible architecture, outsourcing or providing services to others.
Data bindings in a composite application

Modeling, representing and flowing data across components and protocols
A simple scenario

• Two SCA components: Payment and CreditCardPayment (Payment calls CreditCardPayment to authorize credit card charges)
• Two developers: Bob and Mary
• Payment communicates with CreditCardPayment using SOAP/HTTP web service
• The Payment component will be exposed as a JSON-RPC service
The SCA composite file

```xml
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
           xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.1"
           targetNamespace="http://tuscanyscatours.com/" name="Store">

  <component name="Payment">
    <implementation.java class="com.tuscanyscatours.payment.impl.PaymentImpl"/>
    <service name="Payment">
      <tuscany:binding.jsonrpc uri="http://localhost:8080/Payment"/>
    </service>
    <reference name="creditCardPayment">
      <binding.ws uri="http://localhost:8080/CreditCardPayment"/>
    </reference>
  </component>

  <component name="CreditCardPayment">
    <implementation.java class="com.tuscanyscatours.payment.creditcard.impl.CreditCardPaymentImpl"/>
    <service name="CreditCardPayment">
      <binding.ws uri="http://localhost:8080/CreditCardPayment"/>
    </service>
  </component>

</composite>
```
Interface and data modeling

• Bob and Mary first agree on what data needs to be exchanged between the two components
• The agreement is then described as an interface (which in turn references the data types)
  – The interface becomes the key contract between the SCA reference and service that are wired together
  – The interfaces can be described using different IDLs such as WSDL (w/ XSD) or Java interface
    – Interface compatibility
• Things to consider: efficiency, simplicity, remotablity and interoperability.
Sample interfaces

JAX-WS w/ JAXB (annotations omitted...):

```java
public interface CreditCardPayment {
    String authorize(JAXBCreditCardDetailsType creditCard, float amount);
}
```

SDO:

```java
public interface CreditCardPayment {
    String authorize(SDOCreditCardDetailsType creditCard, float amount);
}
```
How are data represented in a composite application?

• Component implementations
  – Business logic needs to consume/produce data in a representation it supports
    • Handling incoming service calls
    • Calling other services
    • Receiving property values
  – Certain implementation containers impose the data representations (such as DOM for Apache ODE BPEL)

• Protocol stacks behind the bindings
  – Protocol stacks need to marshal/unmarshal data
    • Internal data representation
    • Wire format (XML, JSON, binary, etc)
Data representations between two components at runtime
The reality check

• Enforcing one data binding is not flexible or even not feasible
  – Components can be implemented using different technologies which could impose the data binding requirements, for example, a BEPL engine may require DOM
  – Components may choose to use different data bindings to represent the business data (input/output/fault), for example, JAXB vs. SDO for the XML manipulations.

• Service providers or consumers are decoupled and it is impossible to have a fixed data binding
  – A service can serve different consumers that can only handle certain data bindings natively
  – The service providers for a given consumer can be replaced

• The same service can be accessed over different protocols with different data bindings
Data transformation

• Data transformations are required to get two components talk to each other
• Having application code to deal with technological data transformation is a nightmare and it will defeat the whole purpose and promise of SCA
Tuscany’s data binding framework

Introspect/transform data without application coding
What does the framework need to figure out?

• Understand the data binding requirements at different places for the data flow
• Transform the data from one data binding to the other transparently without the interventions from the application developers
• Separate the data transformation/marshaling/unmarshaling from the business logic
Data type introspection

• Marker interface
  – commonj.sdo.DataObject, org.w3c.dom.Node

• Annotations
  – JAXB annotations

• What information is captured?
  – Java class
  – Java generic type
  – Logic type (XML element/type, Mime types, etc)
The magic behind the scenes
Transformation paths

• Types of transformations
  – Unmarshal/Deserialize (InputStream → Object)
  – Marshal/Serialize (Object → OutputStream)
  – Convert/Transform (Object → Object)

• Direct vs. Multi-hops
  – JAXB ↔ DOM
  – JAXB ↔ DOM ↔ SDO
  – Weight of a transformation

• Private vs. Public
  – Some data bindings are not good as intermediaries (data round-trip issues)
The complete data flow

- Customer data come in JSON from HTTP requests
- The JSON data is unmarshaled into JAXB for the Payment code to consume
- Payment passes CreditCard (JAXB) to the reference binding layer which in turn converts JAXB into AXIOM
- The service binding layer unmarshals the XML data into AXIOM and transform it into SDO for the CreditCardPayment
- The response path is reverse for the data transformations
Data bindings out of the box

• DOM
• JAXB (JavaBeans are treated as JAXB)
• SDO
• JSON
• StAX
• ...

Data bindings for RESTful services

Integrating with JAX-RS entity providers
JAX-RS entity providers

• Entity providers supply mapping services between representations and their associated Java types.

• Entity providers come in two flavors:
  – MessageBodyReader
  – MessageBodyWriter
Tuscany’s generic entity providers based on the data binding framework

Entity Providers

JAX-RS runtime (such as Apache Wink)

Tuscany data binding framework

Entity (JAXB, SDO, DOM, etc)

InputStream

OutputStream

Message Reader

Message Writer

OutputStream
Extending the data binding framework
Support your favorite data bindings
The DataBinding SPI

public interface DataBinding {
  String getName();

  boolean introspect(DataType dataType, Operation operation);

  DataType introspect(Object value, Operation operation);

  WrapperHandler getWrapperHandler();

  Object copy(Object object, DataType sourceDataType, DataType targetDataType, Operation sourceOperation, Operation targetOperation);

  XMLTypeHelper getXMLTypeHelper();
}

The Transformer SPI

public interface PullTransformer<S, R> extends Transformer {
    R transform(S source, TransformationContext context);
}

public interface PushTransformer<S, R> extends Transformer {
    void transform(S source, R sink, TransformationContext context);
}
Registering your data bindings/transformers

• META-INF/services/
  org.apache.tuscany.sca.databindinng.DataBinding
    org.apache.tuscany.sca.databindinng.xml.DOMDataBinding;name=org.w3c.dom.Node

• META-INF/services/
  org.apache.tuscany.sca.databindinng.PullTransformer

• META-INF/services/
  org.apache.tuscany.sca.databindinng.PushTransformer
Q&A

Find more details from:

Chapter 9 of Tuscany In SCA Action
http://www.manning.com/laws/

Save 40% at manning.com.
Enter “javaone2010” in the promotional code box at check out.