A GUI for invoking Web Services: benefits and how to build it on top of Apache CXF

Alessio Soldano
alessio.soldano@jboss.com

Principal Software Engineer
JBoss - Red Hat

November 17th, 2016

Let's invoke a

Web Service

in Java...

WSDL reference

localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
-<wsdl:definitions name="OrderMgmtService" targetNamespace="http://retail.advanced.samples.jaxws.ws.test.jboss.org/">
  <wsdl:import location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl=OrderMgmt.wsdl" namespace="http://org.jboss.ws/samples/retail">
  </wsdl:import>
 -<wsdl:binding name="OrderMgmtServiceSoapBinding" type="ns1:OrderMgmt">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   -<wsdl:operation name="prepareOrder">
      <soap:operation soapAction="" style="document"/>
     -<wsdl:input name="prepareOrder">
        <soap:body use="literal"/>
      </wsdl:input>
    -<wsdl:output name="prepareOrderResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
 -<wsdl:service name="OrderMgmtService">
  -<wsdl:port binding="tns:OrderMgmtServiceSoapBinding" name="OrderMgmtBeanPort">
      <soap:address location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

JAX-WS Tools

```
asoldano@localhost /dati/demo-wise/wildfly-11.0.0.Alpha1-SNAPSHOT/bin (master) $ ./wsconsume.sh -k http://localhost:8080/jaxws-samples-retail/Or
derMgmtService/OrderMgmtBean?wsdl
Could not find log4j.xml configuration, logging to console.

Loading FrontEnd jaxws ...
Loading DataBinding jaxb ...
wsdl2java -compile -exsh false -d /dati/demo-wise/wildfly-11.0.0.Alpha1-SNAPSHOT/bin/output -verbose -classdir /dati/demo-wise/wildfly-11.0.0.Al
pha1-SNAPSHOT/bin/output -allowElementReferences http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl
wsdl2java - Apache CXF 3.1.6

asoldano@localhost /dati/demo-wise/wildfly-11.0.0.Alpha1-SNAPSHOT/bin (master) $ ls output/ws/jboss/org/samples/retail/
Customer.class  ObjectFactory.class  OrderItem.class  OrderMgmt.class  OrderState.class  OrderStatus.class  OrderType.class  package-info.class
Customer.java   ObjectFactory.java   OrderItem.java   OrderMgmt.java   OrderState.java   OrderStatus.java   OrderType.java   package-info.java
asoldano@localhost /dati/demo-wise/wildfly-11.0.0.Alpha1-SNAPSHOT/bin (master) $
```

Coding in your IDE

Attività Eclipse mar 16.11

workspace - Java - wise-project-a/src/test/java/org/jboss/wise/demo/WiseProjectATest.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer | Type Hierarchy

- > wise-project-a [demo-wise master]
  - src/main/java
  - target/generated-sources/wsconsume
    - generated.orderMgmtService
      - Customer.java
      - ObjectFactory.java
      - OrderItem.java
      - OrderMgmt.java
      - OrderMgmtService.java
      - OrderState.java
      - OrderStatus.java
      - OrderType.java
      - package-info.java
  - src/test/java
    - org.jboss.wise.demo
      - WiseProjectATest.java
  - src/test/resources
  - JRE System Library [JavaSE-1.8]
  - Maven Dependencies
  - src
  - target
  - jaxws-samples-retail.jar
  - pom.xml
- > wise-project-b [demo-wise master]
- wss4j [wss4j 2.1.x]
- wss4j-bindings [wss4j 2.1.x]
- wss4j-integration [wss4j 2.1.x]
- wss4j-parent [wss4j 2.1.x]
- wss4j-policy [wss4j 2.1.x]

WiseProjectATest.java

```
1  package org.jboss.wise.demo;
2
3  import static org.junit.Assert.*;
12
13 public class WiseProjectATest {
14
15     @Test
16     public void test() {
17         System.out.println("Creating client...");
18         OrderMgmtService service = new OrderMgmtService();
19         OrderMgmt port = service.getOrderMgmtBeanPort();
20
21         System.out.println("Preparing parameters...");
22         OrderType order = new OrderType();
23         Customer customer = new Customer();
24         customer.setCreditCardDetails("1234-4567-890-333");
25         customer.setFirstName("Mario");
26         customer.setLastName("Rossi");
27         order.setCustomer(customer);
28         OrderItem orderItem = new OrderItem();
29         orderItem.setName("first-order");
30         orderItem.setPrice(10.45);
31         order.getItems().add(orderItem);
32         order.setOrderNum(1234);
33
34         System.out.println("Invoking...");
35         OrderStatus orderStatus = port.prepareOrder(order);
36
37         System.out.println("\nInvocation result:");
38         System.out.println("order number   : " + orderStatus.getOrderNum());
39         System.out.println("order status   : " + orderStatus.getStatus());
40         System.out.println("order discount : " + orderStatus.getDiscount());
41
42         assertEquals(1234, orderStatus.getOrderNum());
43         assertEquals("Prepared", orderStatus.getStatus());
44     }
45
46 }
47
```

Maven build automation

```
28      <plugins>
29        <plugin>
30          <groupId>org.jboss.ws.plugins</groupId>
31          <artifactId>maven-jaxws-tools-plugin</artifactId>
32          <version>1.1.1.Final</version>
33          <configuration>
34            <verbose>true</verbose>
35          </configuration>
36          <executions>
37            <execution>
38              <goals>
39                <goal>wsconsume</goal>
40              </goals>
41              <configuration>
42                <wsdls>
43                  <wsdl>http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl</wsdl>
44                </wsdls>
45                <targetPackage>generated.orderMgmtService</targetPackage>
46              </configuration>
47            </execution>
48          </executions>
49        </plugin>
50      </plugins>
51    </build>
```

Satisfied?

- User code bound to generated classes
- Good solution for clients of stable endpoints
- Unconvenient for quick testing of multiple/different endpoints

JBoss Wise

- Java library for easily invoking webservices
- JBoss community project

JBossDeveloper

- Built on top of Apache CXF JAX-WS tooling

JBoss Wise

- Effective client-server decoupling
- Easy browsing of WSDL models
- (nearly) zero-code invocation of WS operations
- Lowering technical entry level to WS testing

Dynamic client API

```
WSDynamicClient client = WSDynamicClientFactory.getJAXWSClientBuilder().keepSource(true).wsdlURL(WSDL_ADDRESS).build();
WSMethod method = client.getWSMethod("OrderMgmtService", "OrderMgmtBeanPort", "prepareOrder");
//...
Map<String, Object> reqMap = new HashMap<String, Object>();
reqMap.put("prepareOrder", orderObject);
InvocationResult invRes = method.invoke(reqMap);
```

localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
-<wsdl:definitions name="OrderMgmtService" targetNamespace="http://retail.advanced.samples.jaxws.ws.test.jboss.org/">
  <wsdl:import location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl=OrderMgmt.wsdl" namespace="http://org.jboss.ws/samples/retail">
  </wsdl:import>
 -<wsdl:binding name="OrderMgmtServiceSoapBinding" type="ns1:OrderMgmt">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  -<wsdl:operation name="prepareOrder">
      <soap:operation soapAction="" style="document"/>
    -<wsdl:input name="prepareOrder">
        <soap:body use="literal"/>
      </wsdl:input>
    -<wsdl:output name="prepareOrderResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
 -<wsdl:service name="OrderMgmtService">
  -<wsdl:port binding="tns:OrderMgmtServiceSoapBinding" name="OrderMgmtBeanPort">
      <soap:address location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Dynamic client API

```
WSDynamicClient client = WSDynamicClientFactory.getJAXWSClientBuilder().keepSource(true).wsdlURL(WSDL_ADDRESS).build();
WSMethod method = client.getWSMethod("OrderMgmtService", "OrderMgmtBeanPort", "prepareOrder");
//...
Map<String, Object> reqMap = new HashMap<String, Object>();
reqMap.put("prepareOrder", orderObject);
InvocationResult invRes = method.invoke(reqMap);
```

localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<wsdl:definitions name="OrderMgmtService" targetNamespace="http://retail.advanced.samples.jaxws.ws.test.jboss.org/">
  <wsdl:import location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl=OrderMgmt.wsdl" namespace="http://org.jboss.ws/samples/retail">
  </wsdl:import>
 -<wsdl:binding name="OrderMgmtServiceSoapBinding" type="ns1:OrderMgmt">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   -<wsdl:operation name="prepareOrder">
      <soap:operation soapAction="" style="document"/>
     -<wsdl:input name="prepareOrder">
        <soap:body use="literal"/>
      </wsdl:input>
     -<wsdl:output name="prepareOrderResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
 -<wsdl:service name="OrderMgmtService">
   -<wsdl:port binding="tns:OrderMgmtServiceSoapBinding" name="OrderMgmtBeanPort">
      <soap:address location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Dynamic client API

```
WSDynamicClient client = WSDynamicClientFactory.getJAXWSClientBuilder().keepSource(true).wsdlURL(WSDL_ADDRESS).build();
WSMethod method = client.getWSMethod("OrderMgmtService", "OrderMgmtBeanPort", "prepareOrder");
//...
Map<String, Object> reqMap = new HashMap<String, Object>();
reqMap.put("prepareOrder", orderObject);
InvocationResult invRes = method.invoke(reqMap);
```

localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<wsdl:definitions name="OrderMgmtService" targetNamespace="http://retail.advanced.samples.jaxws.ws.test.jboss.org/">
  <wsdl:import location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl=OrderMgmt.wsdl" namespace="http://org.jboss.ws/samples/retail">
  </wsdl:import>
 -<wsdl:binding name="OrderMgmtServiceSoapBinding" type="ns1:OrderMgmt">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   -<wsdl:operation name="prepareOrder">
      <soap:operation soapAction="" style="document"/>
     -<wsdl:input name="prepareOrder">
        <soap:body use="literal"/>
      </wsdl:input>
     -<wsdl:output name="prepareOrderResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
 -<wsdl:service name="OrderMgmtService">
   -<wsdl:port binding="tns:OrderMgmtServiceSoapBinding" name="OrderMgmtBeanPort">
      <soap:address location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Dynamic client API

```
WSDynamicClient client = WSDynamicClientFactory.getJAXWSClientBuilder().keepSource(true).wsdlURL(WSDL_ADDRESS).build();
WSMethod method = client.getWSMethod("OrderMgmtService", "OrderMgmtBeanPort", "prepareOrder");
//...
Map<String, Object> reqMap = new HashMap<String, Object>();
reqMap.put("prepareOrder", orderObject);
InvocationResult invRes = method.invoke(reqMap);
```

localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<wsdl:definitions name="OrderMgmtService" targetNamespace="http://retail.advanced.samples.jaxws.ws.test.jboss.org/">
  <wsdl:import location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl=OrderMgmt.wsdl" namespace="http://org.jboss.ws/samples/retail">
  </wsdl:import>
 -<wsdl:binding name="OrderMgmtServiceSoapBinding" type="ns1:OrderMgmt">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   -<wsdl:operation name="prepareOrder">
      <soap:operation soapAction="" style="document"/>
     -<wsdl:input name="prepareOrder">
        <soap:body use="literal"/>
      </wsdl:input>
     -<wsdl:output name="prepareOrderResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
 -<wsdl:service name="OrderMgmtService">
   -<wsdl:port binding="tns:OrderMgmtServiceSoapBinding" name="OrderMgmtBeanPort">
      <soap:address location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Dynamic client API

```
WSDynamicClient client = WSDynamicClientFactory.getJAXWSClientBuilder().keepSource(true).wsdlURL(WSDL_ADDRESS).build();
WSMethod method = client.getWSMethod("OrderMgmtService", "OrderMgmtBeanPort", "prepareOrder");
//...
Map<String, Object> reqMap = new HashMap<String, Object>();
reqMap.put("prepareOrder", orderObject);
InvocationResult invRes = method.invoke(reqMap);
```

localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<wsdl:definitions name="OrderMgmtService" targetNamespace="http://retail.advanced.samples.jaxws.ws.test.jboss.org/">
  <wsdl:import location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean?wsdl=OrderMgmt.wsdl" namespace="http://org.jboss.ws/samples/retail">
  </wsdl:import>
 -<wsdl:binding name="OrderMgmtServiceSoapBinding" type="ns1:OrderMgmt">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   -<wsdl:operation name="prepareOrder">
      <soap:operation soapAction="" style="document"/>
     -<wsdl:input name="prepareOrder">
        <soap:body use="literal"/>
      </wsdl:input>
     -<wsdl:output name="prepareOrderResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
 -<wsdl:service name="OrderMgmtService">
   -<wsdl:port binding="tns:OrderMgmtServiceSoapBinding" name="OrderMgmtBeanPort">
      <soap:address location="http://localhost:8080/jaxws-samples-retail/OrderMgmtService/OrderMgmtBean"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Model browsing

```
Map<String, WSService> s = client.processServices();
Map<String, WSEndpoint> e = s.values().iterator().next().processEndpoints();
Map<String, WSMethod> m = e.values().iterator().next().getWSMethods();
WSMethod method = m.values().iterator().next();
Map<String, ? extends WebParameter> p = method.getWebParams();
```

Model browsing

```
Map<String, WSService> s = client.processServices();
Map<String, WSEndpoint> e = s.values().iterator().next().processEndpoints();
Map<String, WSMethod> m = e.values().iterator().next().getWSMethods();
WSMethod method = m.values().iterator().next();
Map<String, ? extends WebParameter> p = method.getWebParams();
```

OK, what about parameters?

Model browsing

```
Map<String, WSService> s = client.processServices();
Map<String, WSEndpoint> e = s.values().iterator().next().processEndpoints();
Map<String, WSMethod> m = e.values().iterator().next().getWSMethods();
WSMethod method = m.values().iterator().next();
Map<String, ? extends WebParameter> p = method.getWebParams();
```

OK, what about parameters?

- Structured data for non-trivial endpoints
- invocation map requires Object instances

Parameters

- Get Java type from WebParameter instances
- Use reflection on Wise on-the-fly generated classes to build up desired data

```
WebParameter par = method.getWebParams().get("prepareOrder");
Class<?> orderClass = (Class<?>)par.getType();
Object order = orderClass.newInstance();
orderClass.getMethod("setOrderNum", long.class).invoke(order, 1234);
//...
Map<String, Object> reqMap = new HashMap<String, Object>();
reqMap.put("prepareOrder", order);
InvocationResult invRes = method.invoke(reqMap);
```

Parameters

- Get Java type from WebParameter instances
- Use reflection on Wise on-the-fly generated classes to build up desired data

```
WebParameter par = method.getWebParams().get("prepareOrder");
Class<?> orderClass = (Class<?>)par.getType();
Object order = orderClass.newInstance();
orderClass.getMethod("setOrderNum", long.class).invoke(order, 1234);
//...
Map<String, Object> reqMap = new HashMap<String, Object>();
reqMap.put("prepareOrder", order);
InvocationResult invRes = method.invoke(reqMap);
```

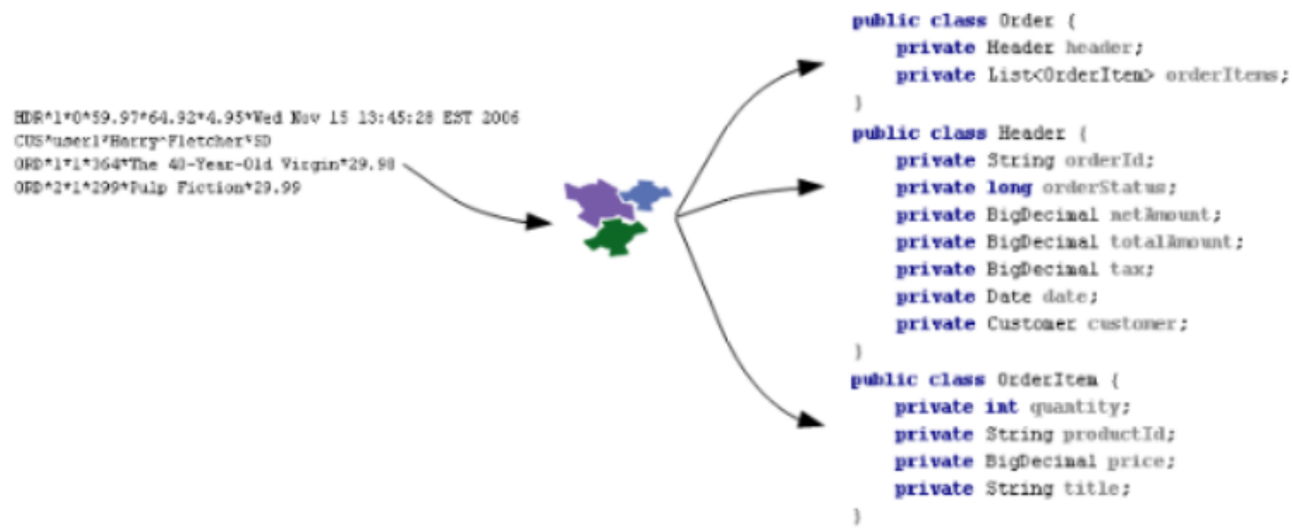
... yes, we can do better :-)

Wise mappers

- Allow users to invoke endpoints using their model
- Map user model to internally generated classes

This turns the problem into **defining the proper transformation** of user data into target endpoint parameters

Smooks mapper



Invoke WS endpoints given **any** model (xml, csv, Java, ...)

Smooks mapper

E.g. define request and response mappings...

smooks-response.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<smooks-resource-list xmlns="http://www.milyn.org/xsd/smooks-1.1.xsd"
    xmlns:jb="http://www.milyn.org/xsd/smooks/javabean-1.2.xsd">
    <params>
        <param name="stream.filter.type">SAX</param>
    </params>
    <jb:bean beanId="StatoOrdine" class="demo.StatoOrdine" createOnElement="$document">
        <jb:value property="numeroOrdine" data="orderNum" />
        <jb:value property="stato" data="status" />
        <jb:value property="sconto" data="discount" />
    </jb:bean>

</smooks-resource-list>
```

... and invoke providing the user model

```java
Ordine ordine = new Ordine();
//...populate custom model
InvocationResult invRes = method.invoke(ordine, new SmooksMapper("smooks-request.xml", client));
StatoOrdine statoOrdine = (StatoOrdine)invRes.getMappedResult(new SmooksMapper("smooks-response.xml", client)).get("StatoOrdine");
```

A generic approach

- Request/response model is the core concept
- We can aim at a simpler and more generic model
- Only String values, maybe?

Tree model

- Tree-like view model
- String only values
- kind of DOM tree built on the valid WS req/res space

```java
package org.jboss.wise.tree;

import java.io.Serializable;
import java.lang.reflect.Type;
import java.util.Iterator;

public interface Element extends Serializable, Cloneable {

    public boolean isLeaf();
    public boolean isRemovable();
    public Type getClassType();
    public String getName();
    public boolean isNil();
    public void setNil(boolean nil);
    public String getId();
    public boolean isNillable();
    public boolean isGroup();
    public void removeChild(String id);
    public Element getChild(String id);
    public Element getChildByName(String name);
    public Iterator<String> getChildrenIDs();
    public Iterator<? extends Element> getChildren();
    public String getValue();
    public void setValue(String value);
    public Element getPrototype();
    public Element incrementChildren();
    public int getChildrenCount();
    public boolean isLazy();
    public boolean isResolved();
    public Element clone();
    public Object toObject();

}
```

Tree model usage

Get the tree view model and populate it..

```
WebParameter par = method.getWebParams().get("prepareOrder");

ElementBuilder builder = ElementBuilderFactory.getElementBuilder().client(client).request(true).useDefautValuesForNullLeaves(true);
Element order = builder.buildTree(par.getType(), par.getName(), null, true);

order.getChildByName("orderNum").setValue("1234");
order.getChildByName("customer").getChildByName("firstName").setValue("Mario");
//...
Element orderItem = order.getChildByName("items").incrementChildren();
orderItem.getChildByName("name").setValue("first-order");
orderItem.getChildByName("price").setValue("10.45");
//...
```

Invoke the endpoint and convert the result into another tree...

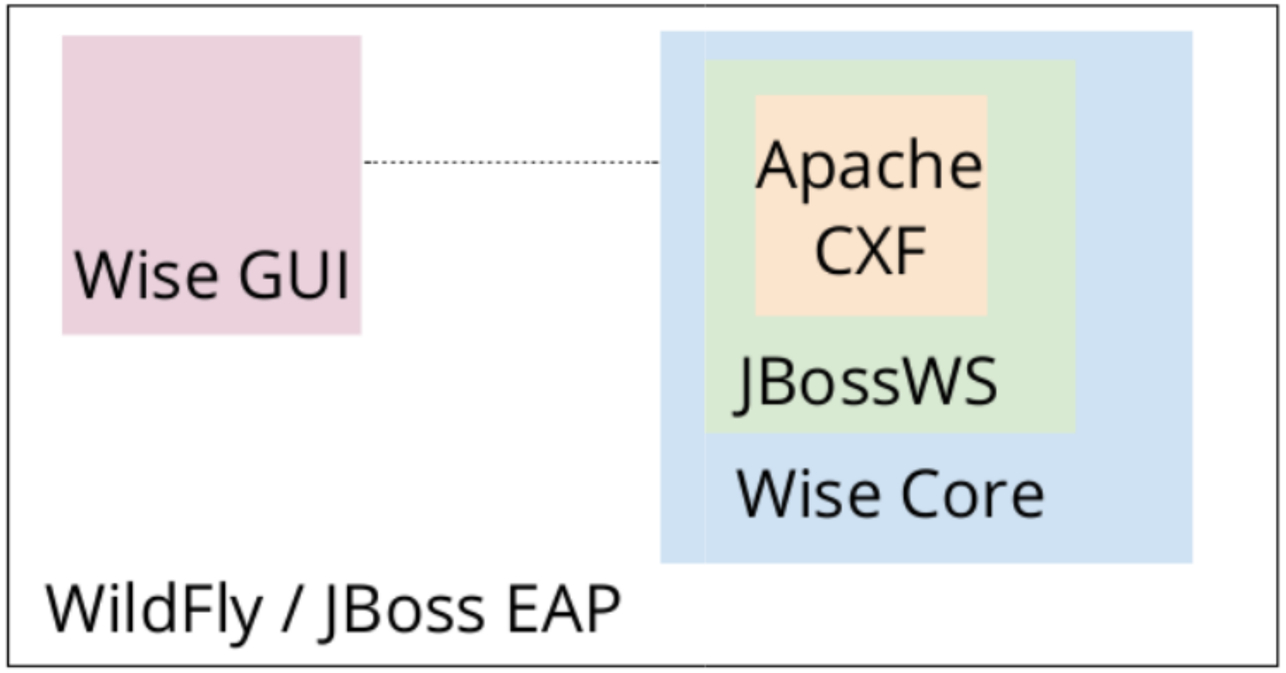
```
Map<String, Object> args = new java.util.HashMap<String, Object>();
args.put(order.getName(), order.toObject());
InvocationResult invRes = method.invoke(args, null);

Class<?> resultClass = (Class<?>)invRes.getResult().get(WSMethod.TYPE_RESULT);
Object result = invRes.getResult().get(WSMethod.RESULT);
Element orderStatus = builder.request(false).useDefautValuesForNullLeaves(false).buildTree(resultClass, "orderStatus", result, true);

for (Iterator<? extends Element> it = orderStatus.getChildren(); it.hasNext(); ) {
    Element el = it.next();
    System.out.println(el.getName() + " : " + el.getValue());
}
```

Tree model advantages

- Detyped model... but still compliant with WSDL/schema
- Simple and generic... enough to build a GUI on it !

Wise GUI

- Default scenario



- But we could also have a fat war to run on any Java EE container

Wise GUI

1. Wise Core uses JAX-WS tooling for generating client classes
2. A tree view is built around the generated client
3. The GUI lets users populate the tree view

- Multiple choices for client side implementation
  - JSF (RichFaces)
  - GWT
  - AngularJS 2 ?

Wise GUI - Benefits

- **Web based**
  - no need for Eclipse / IDE
  - usable everywhere (mobile...?)

- **Focus on the data, not on the technology**
  - No WS, XML or Java knowledge required
  - Fast / agile WS testing
  - Enable business acceptance tests from analysts

Wise Invokes Services Easily

- **Wise is built on top of JAX-WS compliant impl**
  - correctness
  - interoperability
- **You consume WS services based on WSDL/XSD**
  - granted compliance with the contract
  - ...still you don't get your hands dirty with SOAP

Wise Invokes Services Easily

- **Wise is built on top of JAX-WS compliant impl**
  - correctness
  - interoperability
- **You consume WS services based on WSDL/XSD**
  - granted compliance with the contract
  - ...still you don't get your hands dirty with SOAP

... isn't this easy? :-)

wise.jboss.org

github.com/asoldano/demo

Thank you!