EclipseLink: Bean Validation in JAXB
MOXy grows stronger (again)!

Marcel Valovy
TopLink Software Engineer
April 14, 2015
Program Agenda

- EclipseLink
- History
- MOXy
- Bean Validation in JAXB
- Conclusion
- References
EclipseLink Overview
EclipseLink

Advanced object-persistence and object-transformation project

<table>
<thead>
<tr>
<th>Object-Relational Mapping</th>
<th>Object-XML Binding</th>
<th>Database Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPA</td>
<td>JAXB, SDO</td>
<td>JAX-WS, JAX-RS</td>
</tr>
</tbody>
</table>
History of EclipseLink
History ‘90s

1990s The Object People
Consulting company
Research in ORM during 1990s

1994 - Toplink for SmallTalk
History ‘96

1996 TOP team started developments for Java™

1997 TopLink for Java 1.0, written from scratch
History ‘99

1999 “era of J2EE application servers”

Mainstream Adoption:

“TopLink for WebLogic”

“TopLink for WebSphere”
History ‘99

1999 Hot & Award winning product
“The Hottest and Coolest Product of ‘99”

Java Pro, Java World and JDJ “Readers’ Choice Award”
History ‘Millenium

April 2000 - TopLink acquired by startup WebGain

TopLink crew grew from 30 to 90 people

One “TopLink” product (merge of “TopLink for: WebLogic, WebSphere, BlueStone/HP-AS”)
History ‘02

**June 2002** - Oracle acquired TopLink with all its 90 people

“Industry leading persistence layer”

O-R mapping extended with O-X mapping
History ‘07

**2007** - Oracle contributed TopLink 11g source code to Eclipse Foundation

EclipseLink 1.0 was born!
MOXy
MOXy - Overview
MOXy Process Flow

XML schema → Binding compiler → Schema-derived classes & interfaces

XML document → MOXy (JAXB) → Content objects

MOXy (JAXB) → Unmarshal → Application

MOXy (JAXB) → Marshal → XML document
Bean Validation in JAXB
BV in JAXB - Goals

Automatic **runtime validation** for:
- Objects constructed from unmarshalled documents
- Objects entering marshalling process

**Tooltime metadata generation:**
- Generate **BV annotations** in XJC compiled Java classes
- Generate **XML Facets** in SchemaGen generated Schemas
Usage 1 - Unmarshaller

JAXBContext context = JAXBContext.newInstance(…)
Unmarshaller unmarshaller = context.createUnmarshaller()

Employee employee = (Employee) unmarshaller.unmarshal(file)
assert unmarshaller.getConstraintViolations().isEmpty()

No configuration :-(
Usage 2 - Marshaller, handling constraint violations

JAXBContext context = JAXBContext.newInstance(...)  
Marshaller marshaller = context.createMarshaller()  
try {
    marshaller.marshal(invalidEmployee, file)
} catch (BeanValidationException bve) {
    ... business logic handling invalid objects
}
Usage 3 - XJC: Bean Validation Plugin

**Command Line**

```
xjc file.xsd -XBeanVal
```

Example usage with mods:

```
xjc file.xsd -XBeanVal jsr303 simpleRegex
```

**Programmatically**

```java
Driver.run ( new String [ ] { schemaPath, 
    "-extension", "-XBeanVal" }, System.
    out, System.out )
```
Usage 4 - SchemaGen

Map props = new HashMap()
props.put("eclipselink.beanvalidation.facets", true)

JAXBContext jc = JAXBContext.newInstance(classes, properties)

SchemaOutputResolver sor = new MSOR()
jc.generateSchema(sor)

private class MSOR extends SchemaOutputResolver {

    public Result createOutput(String namespaceURI,
                                String suggestedFileName) throws IOException {

        File file = new File(suggestedFileName);
        StreamResult result = new StreamResult(file);
        result.setSystemId(file.toURI().toURL().toString());
        return result;
    }

}
Excellent Performance

**Extensively fine-tuned**

Green = percentage boost achieved

<table>
<thead>
<tr>
<th><code>..testWorkOrderMarshal</code></th>
<th>thrpt</th>
<th>1</th>
<th>50</th>
<th>22452.601</th>
<th>76.6456</th>
<th>ops/s</th>
<th>(64.39 / 0.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>..erOutputStreamMarshal</code></td>
<td>thrpt</td>
<td>1</td>
<td>50</td>
<td>34695.3601</td>
<td>102.8361</td>
<td>ops/s</td>
<td>(82.82 / 0.0)</td>
</tr>
<tr>
<td><code>..OrderResponseMarshal</code></td>
<td>thrpt</td>
<td>1</td>
<td>50</td>
<td>23666.7424</td>
<td>72.323</td>
<td>ops/s</td>
<td>(57.02 / 0.0)</td>
</tr>
<tr>
<td><code>..reOutputStreamMarshal</code></td>
<td>thrpt</td>
<td>1</td>
<td>50</td>
<td>39340.0181</td>
<td>128.3776</td>
<td>ops/s</td>
<td>(92.08 / 0.0)</td>
</tr>
</tbody>
</table>
MOXy Runtime Enhancements - Class Diagram

- Simple API
- Modular Code
- Concurrency Support
- High Performance
- Customizations
Customization

Easy to use, pass **key:value** to method `.setProperty()` on either JAXBContext, Marshaller or Unmarshaller

| BEAN_VALIDATION_MODE                          | ON, AUTO, OFF |
| BEAN_VALIDATION_FACTORY                      | custom        |
| BEAN_VALIDATION_GROUPS                       | custom        |
| GENERATE_FACETS                              | boolean       |
| BEAN_VALIDATION_NO_OPTIMISATION              | boolean       |
Conclusion
Conclusion

Bean Validation in JAXB enables developing safe applications with:

- automatic runtime verification of constraints inside the marshalling framework
- automatic generation of Bean Validation metadata
- automatic generation of XML Facets
- translating between XML Facets and Bean Validation
Thanks for Your attention!

Marcel Valovy
TopLink Software Engineer
April 14, 2015
References