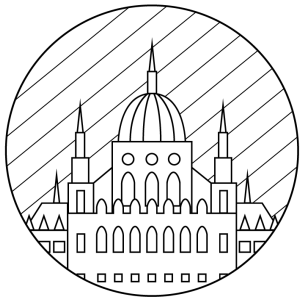
Building the Web Services stack of a JavaEE compliant server around Apache CXF

Alessio Soldano



APACHECON EUROPE

CORINTHIA HOTEL
BUDAPEST, HUNGARY
– OCTOBER 1-3, 2015 –

Agenda

1. History of a collaboration in the open source world
2. Hot integration topics
3. Performances

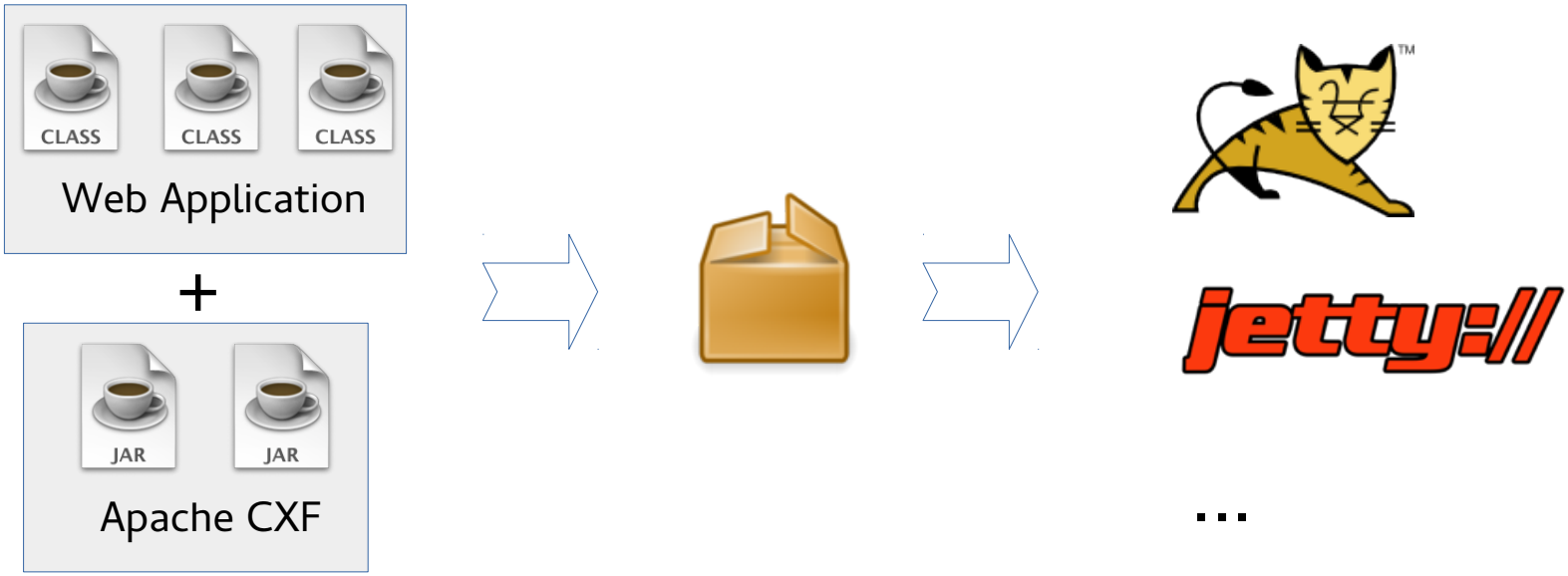
History of a collaboration

Apache CXF Overview

- Apache CXF is a framework for web services (WS) development
  - JAX-WS and JAX-RS programming APIs
  - Multiple transports (HTTP, JMS, ...)
  - Multiple protocols (SOAP, XML/HTTP, RESTfull HTTP, CORBA, ...)
  - Implementation for many WS-* specs (policy, addressing, security, reliable messaging, ...)

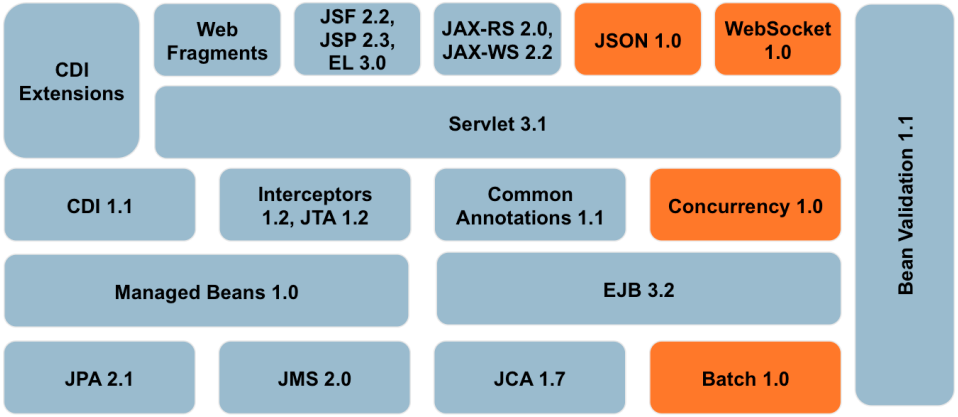
Apache CXF Overview

Common runtime scenario



WildFly / JBoss Application Server

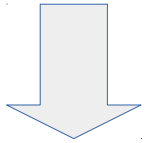
- WildFly is a flexible, lightweight, managed application runtime
  - *Java EE 7 container*
  - *Management*
  - *Modular classloading*
  - *...*



- Provides users with Web Services functionalities according to the JavaEE requirements (JSR 224, JSR 109)
- Allows exposing EE components (POJO, EJB3) as Web Services endpoints

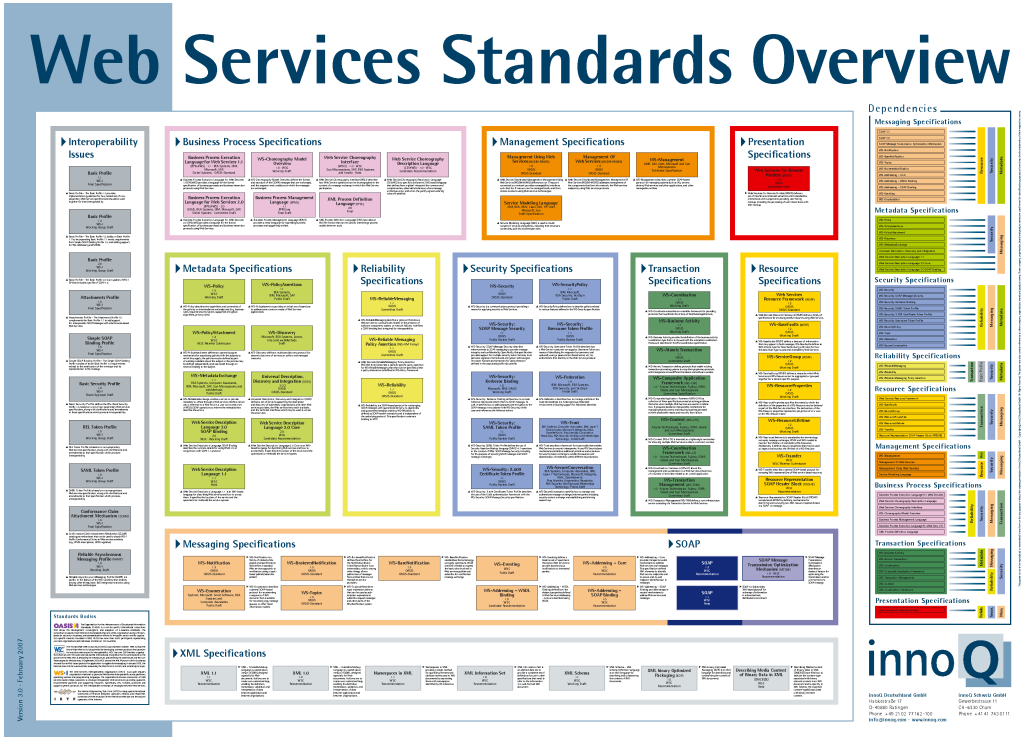
WildFly / JBoss Application Server

- Core of Red Hat Enterprise Application Platform (EAP) product, which Red Hat provides subscriptions and support on.



RED HAT® JBOSS®
ENTERPRISE
APPLICATION PLATFORM

Building a JavaEE server WS stack



Building from scratch...

- *takes way too much time (time to market)*
- *requires a skilled and decently sized team ($$)*
- *the result is likely not worth the efforts.*

Reasons for integrating

- Do not reinvent the wheel
  - *Provide new features instead of wasting time on stuff for which there's already a valid solution*
- Properly allocate energies / resources
  - *Join and help an existing open source project instead of competing with it*
  - *Make the difference on WildFly usage scenarios*
- Merge two communities' user base

Choosing a stack

- Initial prototype phase, multiple integration solutions...
- Technical analysis
  - features, quality of code, performances, extensibility, ...
  - validation through common testsuite and certification tools
- Community analysis
  - size and activity
  - openness to contribution / involvement
- Strategic analysis
  - driving forces behind the project
  - risk assessment on loosing sources control

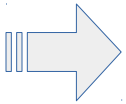
Trust and Contribution

- Trust based on contribution
  - Individual contributors – no Red Hat deal
  - Commit rights and PMC membership gained through contribution
- Trust because of honesty
  - Always keep other parties' needs into consideration – no regressions
  - No commit is for Red Hat, yet for a usecase that's valid regardless of Red Hat will

Issues and Conflicts

- Spring not acceptable as part of the integration
  - We could not distribute Spring in JBoss AS / WildFly
  - Need for encouraging JAX-WS deployment model
  - Internally, useless layer of indirection
- We preferred and fostered the API approach for configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       xmlns:cxf="http://cxf.apache.org/core"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
        http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

    <jaxws:endpoint name="{http://apache.org/hello_world_soap_http}SoapPort"
                    wsdlLocation="wsdl/CustomerService.wsdl" createdFromAPI="true"/>
</beans>
```



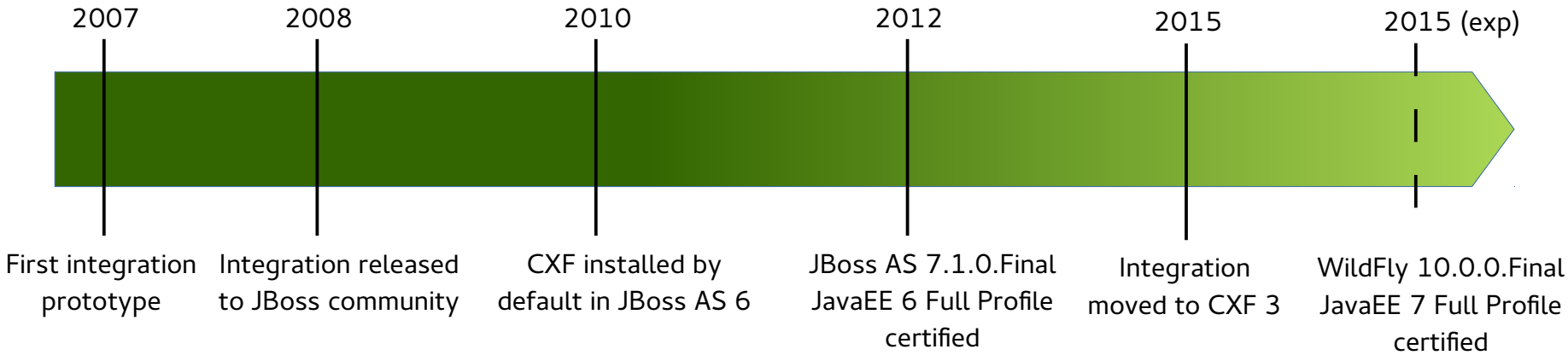
```
import javax.xml.namespace.QName;
import org.apache.cxf.Bus;
import org.apache.cxf.BusFactory;
import org.apache.cxf.jaxws.EndpointImpl;

Bus bus = BusFactory.newInstance().createBus();
Object endpointInstance;
//endpointInstance = new Foo(...)

EndpointImpl endpoint = new EndpointImpl(bus, endpointInstance);
endpoint.setEndpointName(new QName(
        "http://apache.org/hello_world_soap_http", "SoapPort"));
endpoint.setWsdlLocation("wsdl/CustomerService.wsdl");
//...
endpoint.publish();
```

Issues and Conflicts

Still we had issues running without Spring...

- We couldn't drop Spring dependencies from CXF...
- But we could make it optional at runtime:
  - Isolate classes using Spring (CXFServlet vs CXFNonSpringServlet)
  - No direct (or transitive...) Spring import unless it was really needed (and in most cases it wasn't ;-))
- Project lead and other committers agreed on the importance of the changes despite not sharing the immediate need for them
- As a result **any** CXF integrator can pull most of the CXF components without being forced to rely on Spring.

8 Years Balance



- Additional testing scenarios
- JavaEE TCK testing
- Multiple bug and performance fixes

WildFly

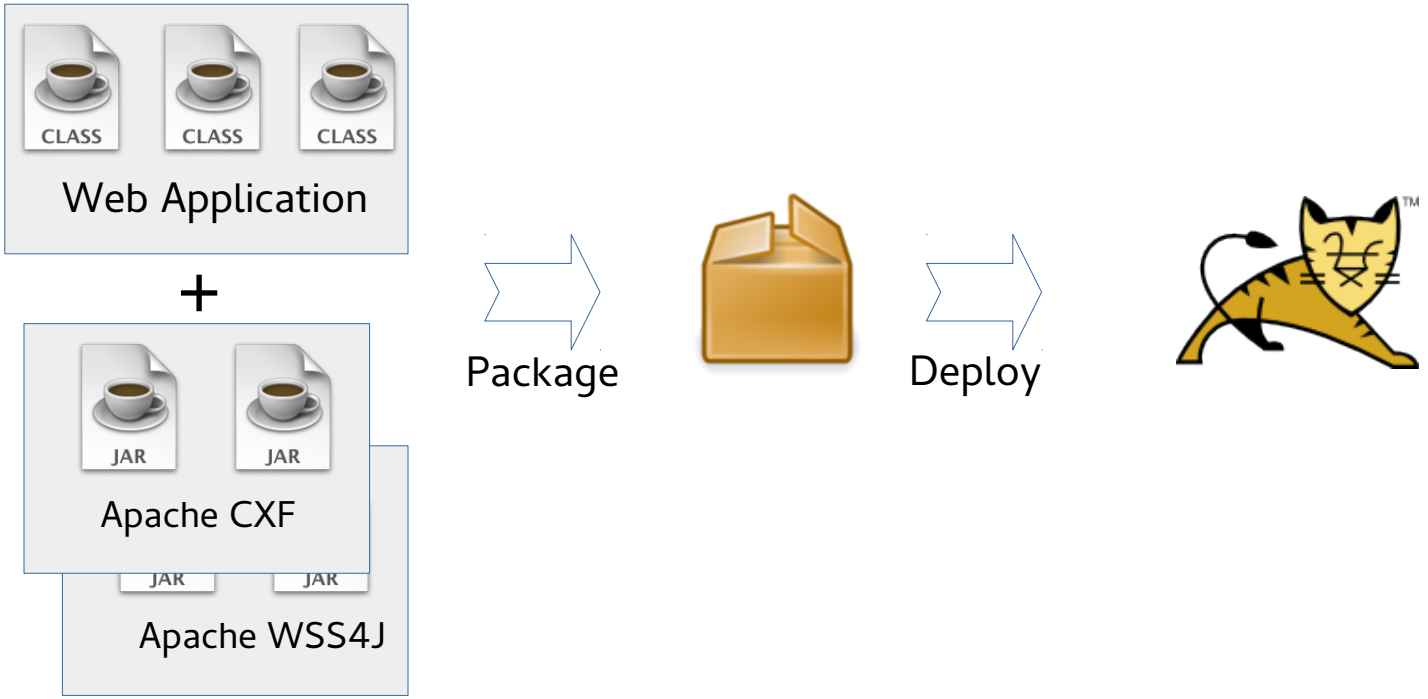
Apache CXF

- Multiple features (including WS-*)
- Multiple bug fixes

Hot integration topics

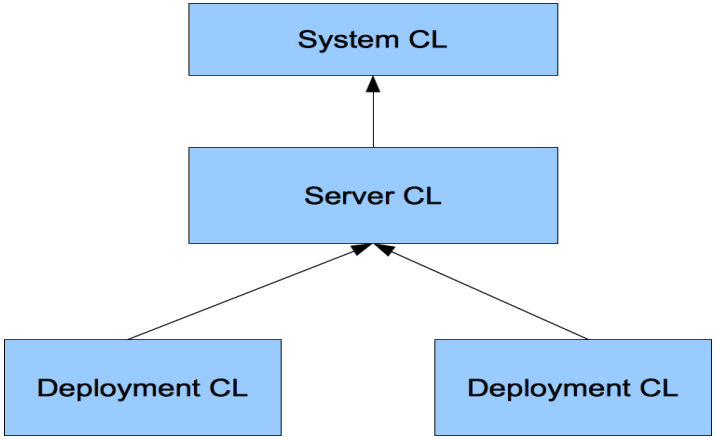
Classloading & deployment

- Deployment of an Apache CXF based web app on Tomcat



Classloading & deployment

- *Hierarchical* classloading



"old" JBoss AS 4.x, 5.x, 6.x

- The *thread context* classloader

  Thread.currentThread().getContextClassLoader()

  is usually set to the web app (defining) classloader

  MyServlet.class.getClassLoader()

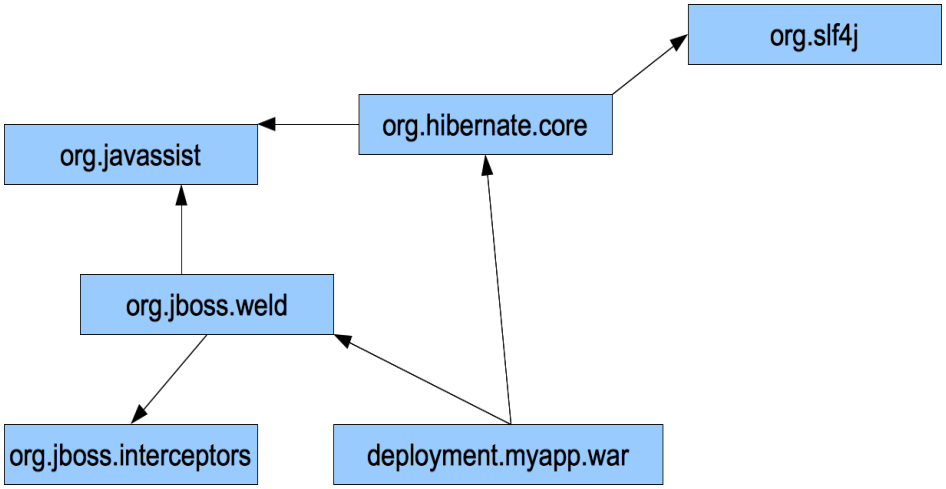
- Web app classes have visibility on almost all server internal classes

Classloading & deployment

- WildFly integration goals
  - Put CXF jars and its dependencies once within the server (not in deployments!)
  - Do not expose deployments to CXF and its dependencies by default (JAX-WS only by default)
  - Let users optionally have visibility over CXF (only, no dependencies)

Classloading & deployment

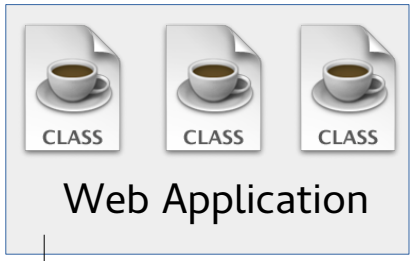
- *Modular, isolated* classloading



- *Modules* define classloader dependencies and class visibility
- Each deployment has its isolated classloader with no visibility over server internals unless dependencies are explicitly set
- The deployment classloader is usually set as thread context classloader
- “OSGI-like”

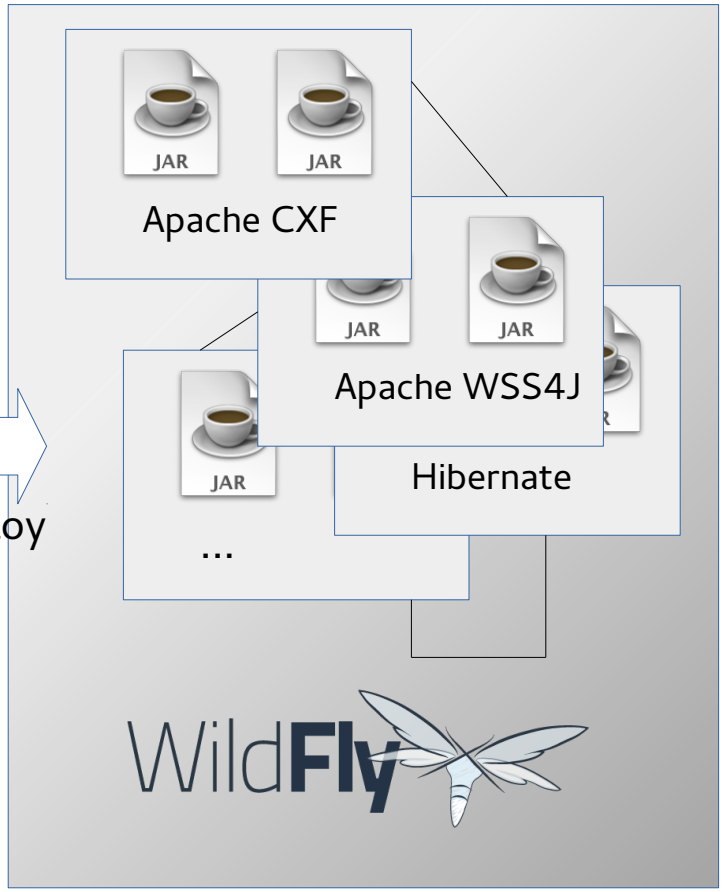
Classloading & deployment

- Deployment of a JavaEE WS app on WildFly



Package

Deploy



MANIFEST.MF
Dependencies: org.apache.cxf

Optional

Classloading & deployment

- A different usage scenario for CXF
  - Not embedded with the application
  - Bus started by WildFly deployment processor, not by the CXFServlet
- We had to
  - **Ensure CXF properly uses its defining classloader when looking for its components and linking to its dependencies**
  - **Ensure CXF relies on the thread context classloader too when looking for user provided (web app) classes**
  - **Set a custom thread context classloader just before hooking into CXF whenever it expects to find both its and users' classes there.**

Bus lifecycle in a JavaEE container

- The Bus is the CXF backbone, containing and providing resources (including endpoints and clients)
  - Each *cxf.xml* descriptor defines a Bus instance
  - Each web app gets its own Bus instance created
- **Default Bus**
  - static reference in BusFactory
  - One instance for each process (better, for each CXF defining classloader...)
- **Thread default Bus**
  - Each Bus instance is linked to the thread that created it
  - No implicit Bus creation if a thread already has its default Bus

Bus lifecycle in a JavaEE container

- JAX-WS clients

```
1  QName serviceName = new QName("http://example.jaxws.ws.test.jboss.org/", "EndpointService");
2  Service service = Service.create(new URL("http://localhost:8080/Foo/Service?wsdl"), serviceName);
3  ServiceType proxy = service.getPort(ServiceType.class);
4  Response response = proxy.getCountryCodes();
```

  - internally need a Bus instance
  - by default, each time you create a Bus you either reuse a thread default Bus (or perhaps even the default Bus) or you create a new Bus which is attached to the current thread.
- In WildFly container:
  - clients usually run by a thread from a thread pool
  - the thread might have previously served a different application!

  - There's a single CXF defining classloader and hence a single default Bus!
  - We can't ask plain JAX-WS users to clean up the Bus links

Bus lifecycle in a JavaEE container

- Integration solution
  - *Never set or rely on the single default Bus*
  - *Custom JAX-WS SPI provider for controlling Bus creation*
  - *Bus creation strategies for JAX-WS clients*
- **Strategies**
  - *Thread bus strategy* (vanilla CXF behavior)
  - *New bus strategy* (always create a new Bus instance)
  - *Thread context classloader strategy* (a Bus instance for each application classloader)
  - Strategies selected according to sys props and/or JAX-WS Features

BouncyCastle global installation

- The problem
  - Crypto algorithms' implementation is provided by security providers installed in the JDK
  - Some algorithms used by CXF/WSS4J are not available with the default set of providers of some JDKs
  - Hence *WSS4J installs BouncyCaste (BC) security provider by default*
  - **This affects anything running in the same JVM**
  - Sooner or later somebody complains (WildFly web layer in our case...)

BouncyCastle global installation

- “Abuse of power” by CXF and WSS4J
- We modified CXF / WSS4J to allow them using BC without affecting others
  - Extended WSS4J with a *ThreadLocalSecurityProvider* delegating to a thread-local *configurable* security provider
  - Installed it at WSS4J boot instead of the BC security provider
  - Had the integration code store a custom (BC) security provider in CXF Exchange
  - Had CXF set the security provider from the Exchange into the *ThreadLocalSecurityProvider*
- *CXF can now effectively use any user-provided security provider without side effects*

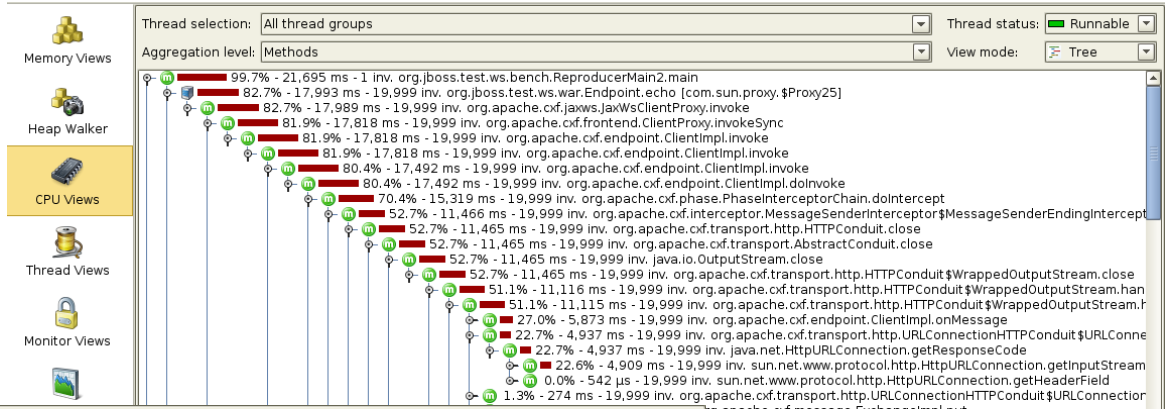
Performances

Benchmarking and Profiling

- No micro-bench
- Full JAX-WS client-server tests
- Multiple scenarios, different payloads, ...
- JDK 7 (JDK 8 runs coming soon)
- Optimization efforts driven by product requirements
  - No regressions among product versions
  - SPECjEnterprise2010

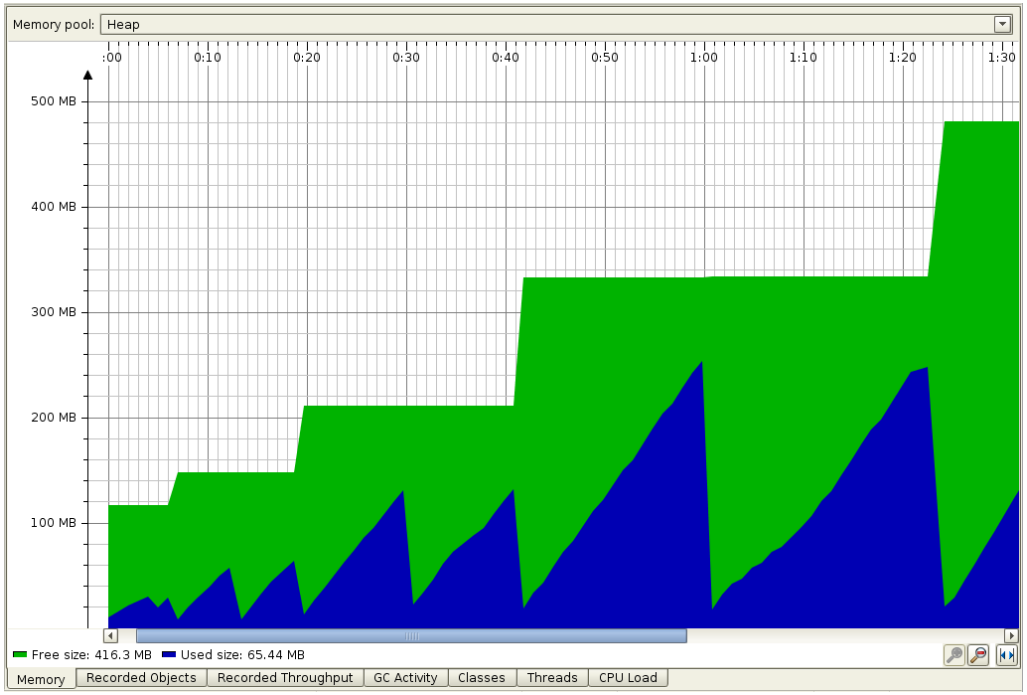
Benchmarking and Profiling

- CPU analysis
  - Tree
  - Hotspots



Benchmarking and Profiling

- Memory analysis



Logs and Strings

- Strings are immutable; the JVM spends a lot of time building and merging them, as new memory arrays have to be allocated
  - consider using *StringBuilder.append(..)* instead of "+"
  - ... but beware the HotSpot already does that!

```
int n = 4;

String cn = "a" + n + "b" + n; //OK

StringBuilder sb1 = new StringBuilder();
sb1.append("a");
sb1.append(n);
sb1.append("b");
sb1.append(n);
String cn1 = sb1.toString(); //OK

StringBuilder sb2 = new StringBuilder();
sb2.append("a" + n); //NO!!
sb2.append("b" + n); //NO!!
String cn2 = sb2.toString();
```

```
String dn = "a" + n + (n > 3 ? "b" + n : "B" + n/2); //NO!!!

StringBuilder sb3 = new StringBuilder();
sb3.append("a");
sb3.append(n);
if (n > 3) {
   sb3.append("b");
   sb3.append(n);
} else {
   sb3.append("B");
   sb3.append(n/2);
}
String dn1 = sb3.toString(); //OK
```

Logs and Strings

- Avoid creating strings any time it's possible
- Beware of logs: strings could be concatenated to later figure out they can't be logged because of logger levels' setup
- On each logging call the logger does a cheap level check

```
LOG.debug("foo");  //OK
if (LOG.isLoggable(Level.FINE)) {
   LOG.debug("foo"); //NO
}

LOG.debug("foo " + n); //NO
if (LOG.isLoggable(Level.FINE)) {
   LOG.debug("foo " + n); //OK
}
```

- Fixed a bunch of these in the code, low cost perf fixes!

Patterns

- Leverage patterns
  - Avoid multiple *String.replace(..)*, *String.replaceAll(..)*, *String.split(..)*
  - Compile frequently used patterns in advance and reuse them instead
  - Relevant in CXF when processing HTTP headers, Cookies, etc.

```
      29  +import java.util.regex.Pattern;
29    30
30    31   public final class HttpHeaderHelper {
           @@ -44,6 +45,7 @@
44    45
45    46       private static Map<String, String> internalHeaders = new HashMap<St
46    47       private static ConcurrentHashMap<String, String> encodings = new Co
      48  +    private static Pattern charsetPattern = Pattern.compile("\"|'");
47    49
           @@ -110,8 +112,7 @@ public static String mapCharset(String enc, String c
110   112            }
111   113            // Charsets can be quoted. But it's quite certain that they car
112   114            // anything like that.
113         -          enc = enc.replace("\"", "").trim();
114         -          enc = enc.replace("'", "");
      115  +          enc = charsetPattern.matcher(enc).replaceAll("").trim();
```

Reflection

- Beware of reflection
  - *Time consuming*
  - *Avoid or reduce usage* if possible

```
77  77      private Class<?>[] getParameterTypes(Method method, Object[] args) {
78  78          Class<?>[] types = method.getParameterTypes();
    79          final Annotation[][] parAnnotations = method.getParameterAnnotations();
79  80          for (int x = 0; x < types.length; x++) {
80              UnwrapParam p = getUnwrapParam(method.getParameterAnnotations()[x]);
    81          UnwrapParam p = getUnwrapParam(parAnnotations[x]);
81  82              if (p != null) {
```

Collections

- Collections
  - *Avoid useless copies of collections* (perhaps because the clients just need the size or the first item, offer specific API methods instead)

el/AbstractMessageContainer.java (+15 -0)

```
216 216     public List<MessagePartInfo> getMessageParts() {
217 217         if (outOfBandParts == null) {
218 218             return new ArrayList<MessagePartInfo>(messageParts.values());
219 219         }
220 220         List<MessagePartInfo> parts = new ArrayList<MessagePartInfo>(messageParts.values());
221 221         parts.addAll(outOfBandParts);
222 222         return parts;
223 223     }
628c953     224     public int getMessagePartsNumber() {
            225         if (outOfBandParts == null) {
            226             return messageParts.size();
            227         }
            228         return messageParts.size() + outOfBandParts.size();
            229     }
            230     public MessagePartInfo getFirstMessagePart() {
            231         if (!messageParts.isEmpty()) {
            232             return messageParts.values().iterator().next();
            233         } else  if (outOfBandParts != null && !outOfBandParts.isEmpty()) {
            234             return outOfBandParts.get(0);
            235         } else {
            236             return null;
            237         }
            238     }
3dc89fe 224 239     public List<MessagePartInfo> getOutOfBandParts() {
        225 240         if (outOfBandParts == null) {
        226 241             return Collections.emptyList();
        227 242         }
        228 243         return Collections.unmodifiableList(outOfBandParts);
        229 244     }
```

3 ...ndings/soap/src/main/java/org/apache/cxf/binding/soap/interceptor/SoapOutInterceptor.java

```
@@ -214,8 +214,7 @@ private boolean handleHeaderPart(boolean preexistingHeaders, SoapMessage
214 214             return endedHeader;
215 215         }
216 216
217     -       List<MessagePartInfo> parts = wrappedBmi.getMessageInfo().getMessageParts();
218     -       if (parts.size() > 0) {
    217 +       if (wrappedBmi.getMessageInfo().getMessagePartsNumber() > 0) {
219 218             MessageContentsList objs = MessageContentsList.getContentsList(message);
220 219             if (objs == null) {
221 220                 return endedHeader;
```

8 rt/bindings/xml/src/main/java/org/apache/cxf/binding/xml/XMLFormatValidator.java

```
@@ -67,8 +67,8 @@ private boolean checkXMLFormat(BindingInfo binding) {
67 67         for (BindingOperationInfo bo : bos) {
68 68             OperationInfo op = binding.getInterface().getOperation(bo.getName());
69 69             needRootNode = false;
70     -           if (op.getInput().getMessageParts().size() == 0
71     -               || op.getInput().getMessageParts().size() > 1) {
    70 +           final int inputPartsNum = op.getInput().getMessagePartsNumber();
    71 +           if (inputPartsNum == 0 || inputPartsNum > 1) {
72 72                 needRootNode = true;
73 73             }
74 74             if (needRootNode) {
```

Collections

- Collections
  - *Consider lazy creation*
  - *Choose proper implementation*
  - *Properly set initial size*, if possible, to avoid re-allocations

DOM vs StAX

- XML processing: streaming (StAX) vs DOM
- CXF needs a DOM tree for processing (optional) SOAP Headers
- Implemented a mechanism for building the DOM tree only if there actually are SOAP Headers

1. Don't build the tree in advance (perhaps there is no header)
2. Store streaming events till you find a header
3. Use stored events to build up the DOM tree if you find a header; then stop collecting new events
4. Otherwise drop stored events
5. Move to SOAP Body processing

URLs

- Building up a URL is very expensive
  - A lot of String processing
  - URLStreamHandler
- CXF needs endpoint addresses in multiple locations as *URI*, *URL* or *String* instances
- **Lazily perform the conversions once for each address value and save the data in a wrapper *Address* object**

Connections

- *HttpURLConnection* selects a *Proxy* unless one is provided
- The default selection mechanism is quite expensive
- Specify **Proxy.NO_PROXY** in *connection.openConnection(..)* when it's early known no proxy is actually to be used.

JDK HttpClient.available()

- https://bugs.openjdk.java.net/browse/JDK-8044726
- Badly affects HTTP client performances since *7u40* if
  - *keepAlive is true* and
  - *connection.setFixedLengthStreamingMode(..) has been called*
- Won't be fixed
- Measured a **600%** perf regression in a JAX-WS CXF client...
- Workaround:

```
                277  277        protected void setFixedLengthStreamingMode(int i) {
e05d813         278               connection.setFixedLengthStreamingMode(i);
ea4a8de              278           // [CXF-6227] do not call connection.setFixedLengthStreamingMode(i)
                     279           // to prevent https://bugs.openjdk.java.net/browse/JDK-8044726
ea4a8de         279  280        }
ea4a8de              281        protected void handleNoOutput() throws IOException {
                     282            if ("POST".equals(getMethod())) {
                     283                connection.getOutputStream().close();
                     284            }
e05d813              285        }
```

Q & A

- THANKS !!

- Q & A