Chariot Solutions

Kicking the Tires on the Bus

ETE

CHARIOT SOLUTIONS

The Problem

- Being asked by clients to assist in evaluating and implementing SOA solutions
- Many products
- Development Environment:
  Traditional (app server under the IDE) vs. SOA (multiple endpoints and communication protocols)
- Vendor tutorials limited
  - “Hello World” BEPL?
  - mvn jetty:run
  - Click this, Drag that

What Does SOA Do?

- The Service
  - Receive a message
  - Take its time processing it
  - Finish successfully (or not)
  - Report results (or not)
- Architecture
  - Respond to “evolving” business use cases

An Approach

- Figure out how to create configurable POJOSs to mimic service endpoints
  - Protocol
  - Performance
  - Behavior
  - Response
- Design a use case
- Implement that use case using various SOA product suites

Protocol interface: Mule

- Reasonably Mature
- Easily Embedded
- Good Documentation
- Lots of Examples
- Straightforward Design

Protocol interface: Mule

```
<model name="basicJMS">
   <mule-descriptor name="jmsComponent" implementation="SuccessfulStandardServiceWithFileResponse">
     <inbound-router>
       <endpoint address="jms:///FileResponseJmsQueueTest" remoteSync="true" connector="jmsConnector" />
     </inbound-router>
   </mule-descriptor>
</model>

<model name="basicWS">
   <mule-descriptor name="pojoFileResponse" implementation="SuccessfulStandardServiceWithFileResponse">
       <inbound-router>
         <endpoint address="axis:http://localhost:65082/services"  remoteSync="true">
            <properties>
               <map name="soapMethods">
                  <property name="processMessage" value="message;string;in,return;string"/>
               </map>
            </properties>
         </endpoint>
       </inbound-router>
   </mule-descriptor>
</model>
```

Everything Else (Spring POJO)

```
<bean id="SuccessfulStandardServiceWithFileResponse"
                        class="com.chariotsolutions.soaservice.service.StandardServiceWithResponseImpl">
        <property name="responseTimer" ref="SimpleResponseTimer"/>
        <property name="archiver" ref="StringToFileArchiver"/>
        <property name="verifier" ref="BooleanVerifier"/>
        <property name="response" ref="FileResponse"/>
</bean>
```

- Performance
- Behavior
- Response

Problem and ... Another Problem

- Problem
  - Spring application needs protocol translation
- Solution
  - Embed Mule
- Problem
  - Application starts a Spring application-context
  - Mule starts a Spring application-context
  - 2 Spring application-contexts are not better than one

Solution: SpringMuleBootstrap

- Solution
  - The SpringMuleBootstrap class (http://mule.mulesource.org/jira/browse/MULE-659)

```
<!-- In your Spring applicationContext.xml -->
<bean id="muleManager" class="org.mule.extras.spring.config.SpringMuleBootstrap">
      <property name="configResources" value="classpath:/../mule-config.xml"/>
</bean>
```

- Nice Because:
  - Configure Mule in Mule
  - Configure Spring in Spring
  - Share between the two

Exit to Demo

Okay, BUT....

(+)

- Learned a lot
- Easy to get going
- Reusable

(-)

- Shallow
- Lacked Visual Appeal

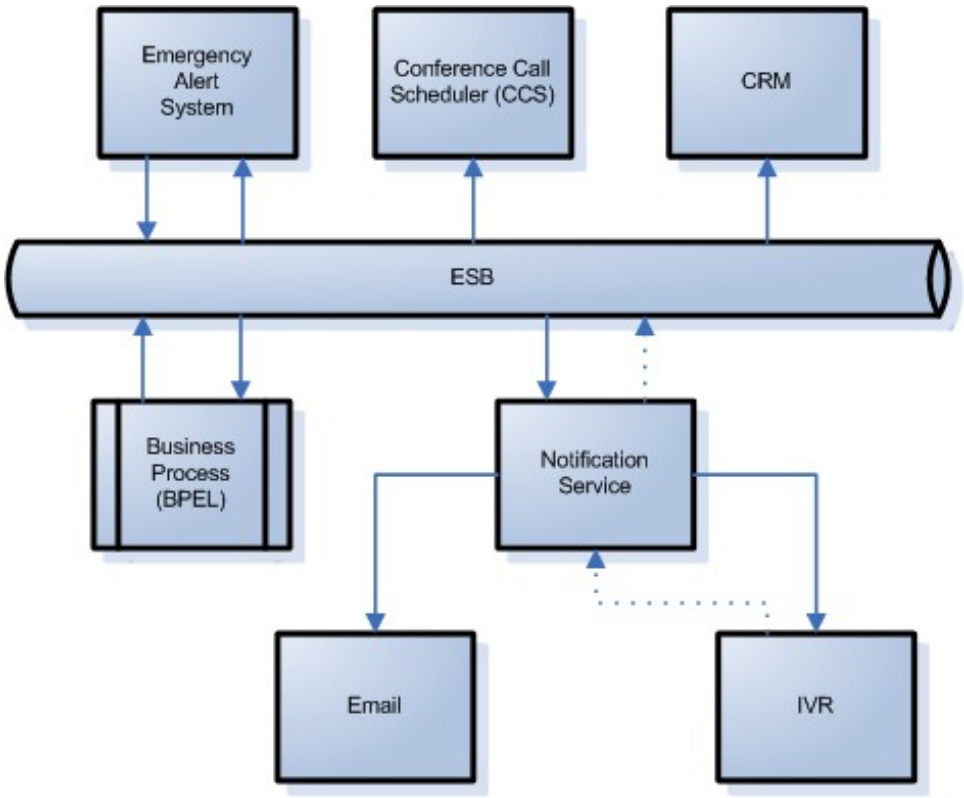
The Solution

- Define a use case
- Build the applications
  - Generic
  - Flexible
  - Lightweight frameworks
  - Small footprint
- Implement

The use case

- Submit an event with the Emergency Alert System (EAS)
- Access CRM System to get a contact list based on the group to contact
- Schedule the conference call with the Conference Call System (CCS)
- EAS updated with the conference call details
- Notify contacts via:
  - Email
  - Interactive Voice Response(IVR)
- EAS updated with the notifications sent
- IVR Notification Response received
- EAS updated with contact response

SOA Lab Endpoint Applications

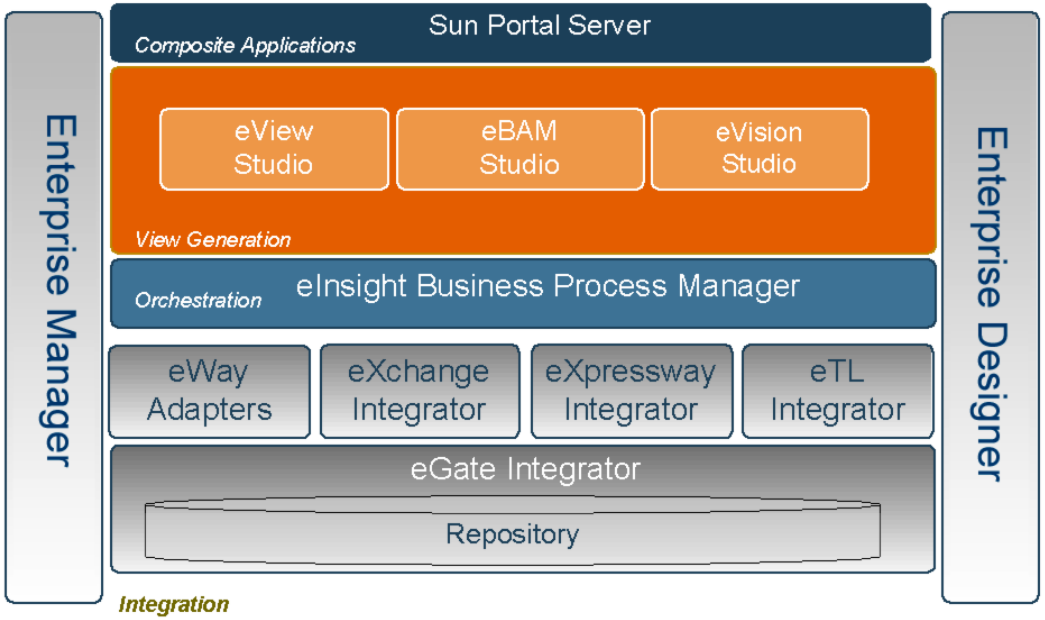


- Emergency Alert System
- CRM System
- Conference Call Scheduler
- IVR System
- Email System
- Notification Service

SOA Lab Endpoint Applications Architecture

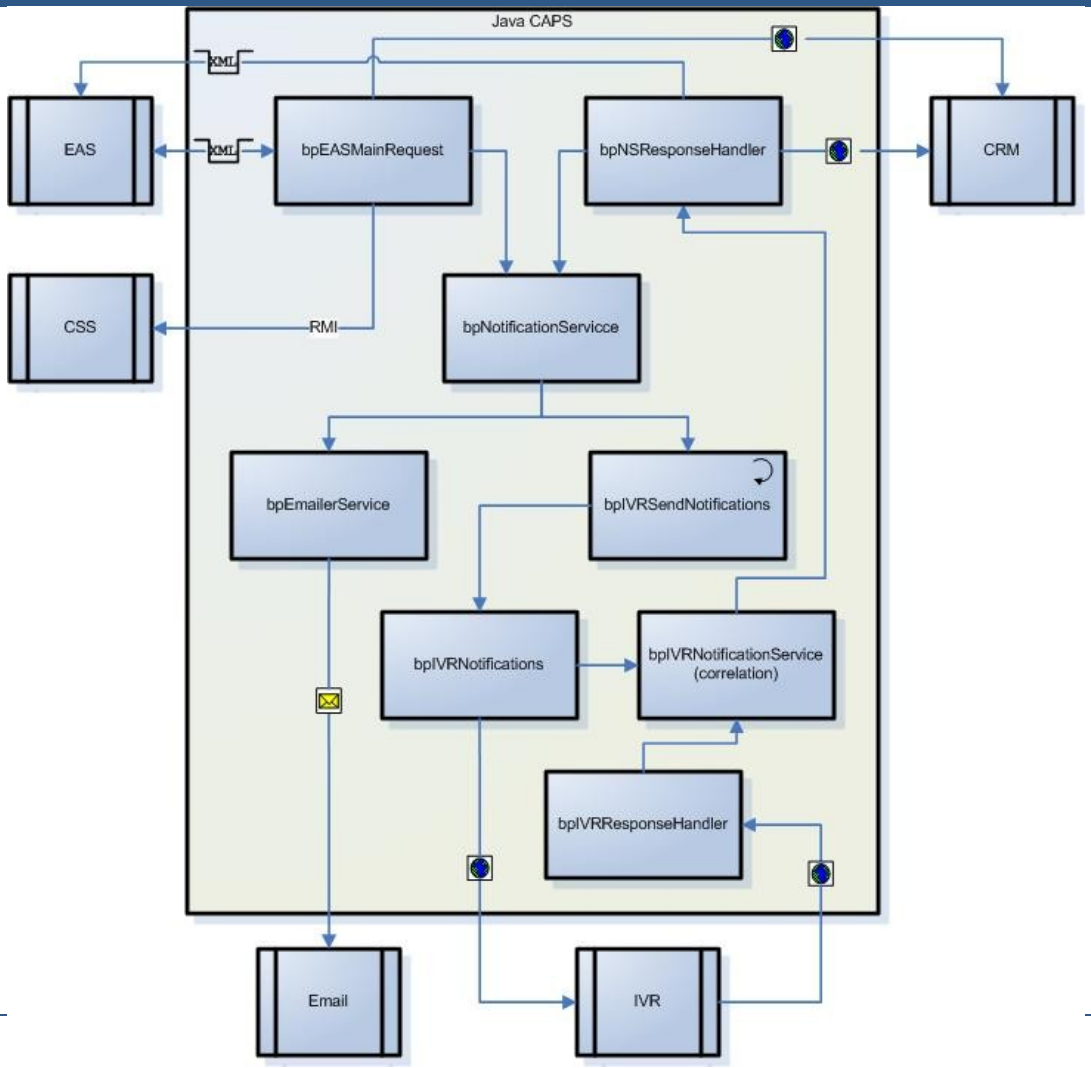
- The endpoint application stack
  - Spring MVC - Web layer
  - Spring - IOC container
  - Mule - Protocol interface
  - Geronimo - Application server
  - ActiveMq - JMS
  - Derby - DB

Sun JCAPS Implementation

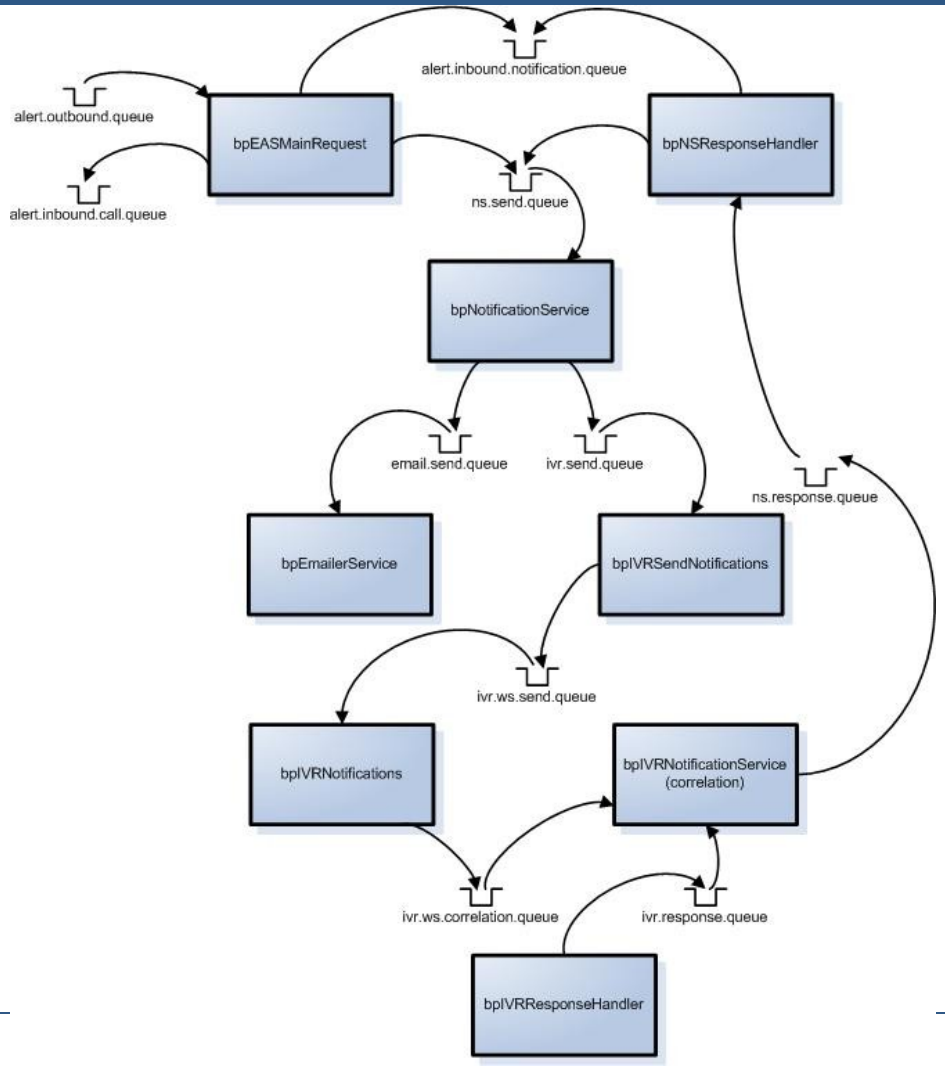


- Java Composite Application Platform Suite
- Formerly SeeBeyond
- The Future is OpenESB

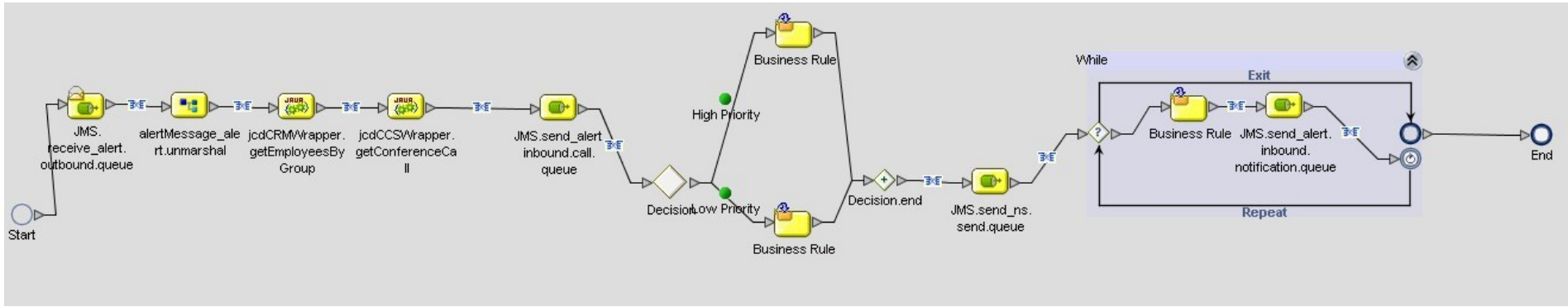
Sun JCAPS: Business Process



Sun JCAPS: Queue Design



Sun JCAPS Implementation



Exit to Demo

The SOA-Lab Pipe Line

- CapeClear
- OpenESB
- ServiceMix
- JBoss
- BEA Aqualogic

Conclusion

- No SOA in a box
- All frameworks require expertise to implement
- SOA needs to be approached top down
  - Establish use cases
  - Evaluate based on those use cases
  - Lab test against the use case
- Your Bus Is Just a Service on Your Bus

Obligatory Self-Promotion

- ME:
  - Email: tpurcell@chariotsolutions.com
  - Blog: http://tompurcellstechblog.blogspot.com/
- My Company:
  - http://www.chariotsolutions.com
- Others:
  - Mule: http://mule.mulesource.org
  - JCAPS: http://www.sun.com/software/javaenterprisesystem/javacaps
  - ServiceMix: http://servicemix.apache.org
  - soapUI: http://www.soapui.org/