JBoss ESB 4.4 GA

Trailblazer Guide

JBESB-TB-8/5/08

Legal Notices

The information contained in this documentation is subject to change without notice.

JBoss Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. JBoss Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Java™ and J2EE is a U.S. trademark of Sun Microsystems, Inc. Microsoft® and Windows NT® are registered trademarks of Microsoft Corporation. Oracle® is a registered U.S. trademark and Oracle9™, Oracle9 Server™ Oracle9 Enterprise Edition™ are trademarks of Oracle Corporation. Unix is used here as a generic term covering all versions of the UNIX® operating system. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Copyright

JBoss, Home of Professional Open Source Copyright 2007, JBoss Inc., and individual contributors as indicated by the @authors tag. All rights reserved.

See the copyright.txt in the distribution for a full listing of individual contributors. This copyrighted material is made available to anyone wishing to use, modify, copy, or redistribute it subject to the terms and conditions of the GNU General Public License, v. 2.0. This program is distributed in the hope that it will be useful, but WITHOUT A WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

See the GNU General Public License for more details. You should have received a copy of the GNU General Public License, v. 2.0 along with this distribution; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Software Version

JBoss ESB 4.4 GA

Restricted Rights Legend

Use, duplication, or disclosure is subject to restrictions as set forth in contract subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause 52.227-FAR14.

© Copyright 2008 JBoss Inc.

Contents

Table of Contents

About This Guide

What This Guide Contains

The Trailblazer Guide contains descriptions on the principles behind Service Oriented Architecture and Enterprise Service Bus, as well as how they relate to JBossESB. This guide also contains information on how to use JBoss ESB 4.4 GA.

Audience

This guide is most relevant to engineers who are responsible for using JBoss ESB 4.4 GA installations and want to know how deploy and test the Trailblazer found under the *Samples*.

Prerequisites

You will need the JBossESB distribution, source or binary to run the trailblazer. You will also need an instance of either the JBoss ESB Server 4.2.1.GA or an instance of JBoss Application Server installed with the JBossESB. The "Getting Started" guide found in the *docs* directory contains instructions on installing and configuring the server.

To test the email notification of quotes, you will require a mail server or the information from your ISP/company email server.

Organization

This guide contains the following chapters:

- **Chapter 1, Overview:** an overview of the loanbroker trailblazer scenario used in JBossESB.
- **Chapter 2, In Depth Look:** a more detailed look at the various artifacts that make up the trailblazer.
- **Chapter 3, Deploying and Testing the TB:** how to compile, deploy, and test the trailblazer.

Documentation Conventions

The following conventions are used in this guide:

Convention	Description
Italic	In paragraph text, italic identifies the titles of documents that are being referenced. When used in conjunction with the Code text described below, italics identify a variable that should be replaced by the user with an actual value.
Bold	Emphasizes items of particular importance.
`Code`	Text that represents programming code.
Function | Function	A path to a function or dialog box within an interface. For example, "Select File | Open." indicates that you should select the Open function from the File menu.
( ) and |	Parentheses enclose optional items in command syntax. The vertical bar separates syntax items in a list of choices. For example, any of the following three items can be entered in this syntax: `persistPolicy (Never | OnTimer | OnUpdate | NoMoreOftenThan)`
Note: **Caution**:	A note highlights important supplemental information. A caution highlights procedures or information that is necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

Table 1 Formatting Conventions

Additional Documentation

In addition to this guide, the following guides are available in the JBoss ESB 4.4 GA documentation set:

1. **JBoss ESB 4.4 GA** *Administration Guide*: How to manage the ESB.
2. **JBoss ESB 4.4 GA** *Getting Started Guide*: Provides a quick start reference to configuring and using the ESB.
3. **JBoss ESB 4.4 GA** *Programmers Guide*: How to use JBossESB.
4. **JBoss ESB 4.4 GA** *Release Notes*: Information on the differences between this release and previous releases.
5. **JBoss ESB 4.4 GA** *Services Guides*: Various documents related to the services available with the ESB.

Contacting Us

Questions or comments about JBoss ESB 4.4 GA should be directed to our support team.

Chapter 1

Trailblazer

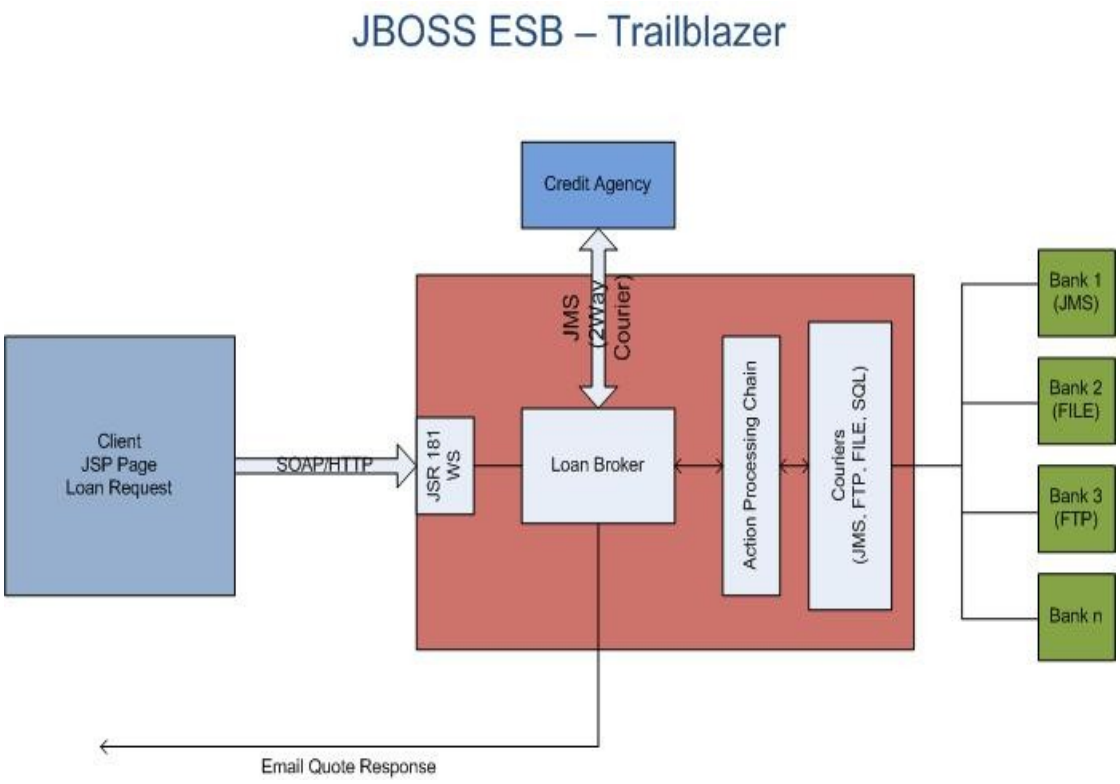
Overview

The Trailblazer is meant to show a commonly understood use-case where the JBossESB can be used to solve the integration problem at hand. The TB is loosely based on the Enterprise Applications Integration book (http://www.eaipatterns.com/). The scenario is very simple - a user is shopping around for a bank loan with the best terms, rate, etc. A loan broker will act as middle-man between the user and the banks. The LoanBroker will gather all the required information from the user, perform a credit check, and then if a suitable credit score is returned, pass the quote request on to each bank. As the quotes are received from the various banks, the LoanBroker will pass those back to the requesting user. This is a common practice in the financial services world today – it's a model used for insurance quotes, mortgage quotes, and so on.

A simple scenario as described above, actually puts forth several integration challenges. Each service (credit agency or bank) has it's own data feed structure (xml, delimited, positional, etc), it's own communication protocol (file, jms, ftp, etc), and finally the responses from each of these is very unique to each. A LoanBroker acting as the agent for these institutions must be able to accommodate each scenario, without expecting the bank or credit agency to adjust anything. The institution's provide a service, and have a clearly defined contract in which to carry out that service. It's our job as the LoanBroker developer to ensure we can be as flexible and adaptable as possible to handle a variety of possible communication protocols, data formats and so on.

This is where JBossESB comes in. Traditionally, an organization would create custom code and scripts to manage the end to end integration between the LoanBroker and each institution (*aka* point-to-point interfaces). This is cumbersome, and messy when it comes to maintenance. For example, adding new banks, and new protocols is not easy. JBossESB gives us a central framework for developing a solution built around a common set of services which can be applied over and over to each unique bank requirement. Adding a new bank then becomes trivial, and support is a lot simpler when you only need to work on one common codebase.

The diagram below shows the scenario at a high level:



* the diagram above is not using any specific notation or style (some of you might be expecting the EIP symbols).

Chapter 2

In Depth Look

Client

The client is a simple JSP page, which routes the submit to a waiting web service. The Loan Request consists of the typical information you would expect from such a request: A social security number (ssn), some personal information like name, address, and so on, as well as loan specific information – loan amount, etc.

Web Service

The web service, which is responsible for receiving the loan requests is a JSR-181 based annotated web service. An annotated web service let's you take any pojo and expose the methods as being capable of receiving requests. The class looks as follows:

```
package org.JBoss.soa.esb.samples.trailblazer.web;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import org.apache.log4j.Logger;
import org.JBoss.soa.esb.samples.trailblazer.loanbroker.LoanBroker;
/**
 * The Loan broker web service, which will handle a loan request.
 */

@WebService(name = "LoanBrokerWS",
targetNamespace = "http://localhost/trailblazer")
@SOAPBinding(style = SOAPBinding.Style.RPC)

public class LoanBrokerBean
{
  private static Logger logger = Logger.getLogger(LoanBrokerWS.class);
  @WebMethod
  // method name is .NET friendly
  public void RequestLoan(WebCustomer customer) {
        logger.info("WebCustomer received: \n" + customer);
        LoanBroker broker = new LoanBroker();
        broker.processLoanRequest(customer);
  }
}
```

JSR-181 annotated pojo web services are a very easy and powerful way to expose plain old java classes as web services. The JBossESB does not have built in support for web services yet, but since we are working in Java, there is no reason why you cannot combine your own web services with the JBossESB services, which is what was done in the trailblazer. The class above is the server side web service. You still need to provide the client, the JSP in this case the client stubs to communicate with the web service. JBossTools which has a web service client side generator to create these classes if you are looking for a tool to use for this.

The most important piece in the web service, is the line which invokes the LoanBroker object, and passes a customer object for processing.

LoanBroker/ESB Components

The Loan Broker is a standard java application, which makes use of the services available in the JBossESB to get data to and from the credit agency and banks, and then finally back to the customer as an email response using the notifier service.

Let's look first at the ESB components required for processing a loan request.

In this release, the Credit Agency is an "ESB Aware" service that requires CSV (Comma Separated Value) information, and the Banks bundled include a JMS based bank and a File based bank, both using XML formatted information. The banks are implemented as external entities. In this example, we are using smooks to do the transformation job, such as CSV to XML, or XML to CSV. (Transformation was described much detail on Message Transofrmation Guide).

In a real production world scenario, these might be internal systems, accessible within your own network, or they may be external providers which you will need to communicate with through some protocol. Needless to say, for this example, we are not focusing on aspects like security, authentication, and other concerns which you would most certainly face. We are focusing solely on the JBossESB components and some sample configurations which you could use to create a similar scenario.

JBossESB has a concept of "Gateway" and "ESB Aware" services. ESB Aware services are able to communicate with the ESB directly using native APIs found in the ESB. These APIs for instance require that you use a Message object. Since the LoanBroker is java based, and has access to the JBossESB APIs, it will be an ESB Aware service. The banks on the other hand, are NON-ESB Aware services. They have no idea, nor should they know anything about the ESB. It is the job of the services in the ESB to facilitate communication to and from the banks, as well as data transformation to/from and so on. These services (the Banks) will interact with the JBossESB through what we call Gateway Services. To read more on the differences between the two, please see the Programmer's Guide.

Let's look at just how you configure the various services in JBossESB. Inside the <TRAILBLAZER_ROOT>/esb/conf/jboss-esb.xml you will see the following deployed services:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<jbossesb
xmlns="http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/etc/schemas/xml/jbossesb-1.0.1.x
sd"
                        parameterReloadSecs="5000">

      <providers>
            <jms-provider name="JBossMQ" connection-factory="ConnectionFactory"
                        jndi-context-factory="org.jnp.interfaces.NamingContextFactory"
                              jndi-URL="localhost">
                  <jms-bus busid="creditAgencyRequest">
                        <jms-message-filter dest-type="QUEUE"
                                    dest-name="queue/esb-tb-creditAgencyQueue"
                                    selector="function='request'"/>
                  </jms-bus>
                  <jms-bus busid="jmsBankRequest">
                        <jms-message-filter dest-type="QUEUE"
                                    dest-name="queue/esb-tb-jmsBankRequestQueue"/>
```

```
            </jms-bus>
            <jms-bus busid="fileBankRequest">
                  <jms-message-filter dest-type="QUEUE"
                              dest-name="queue/esb-tb-fileBankRequestQueue"/>
            </jms-bus>
            <jms-bus busid="jmsBankResponseGateway">
                  <jms-message-filter dest-type="QUEUE"
                              dest-name="queue/esb-tb-jmsBankGatewayResponseQueue"/>
            </jms-bus>
            <jms-bus busid="jmsBankResponseListener">
                  <jms-message-filter dest-type="QUEUE"
                              dest-name="queue/esb-tb-jmsBankResponseQueue"/>
            </jms-bus>
            <jms-bus busid="fileBankResponseListener">
                  <jms-message-filter dest-type="QUEUE"
                              dest-name="queue/esb-tb-fileBankResponseQueue"/>
            </jms-bus>
            <jms-bus busid="customerNotifier">
                  <jms-message-filter dest-type="QUEUE"
                              dest-name="queue/esb-tb-customerNotifier"/>
            </jms-bus>
      </jms-provider>

      <fs-provider name="FSprovider1">
            <fs-bus busid="fileBankChannel" >
                  <fs-message-filter
                        directory="/tmp/output"
                        input-suffix=".quote"
                        work-suffix=".esbWorking"
                        post-delete="true"
                        error-delete="true" />
            </fs-bus>
      </fs-provider>
</providers>
<services>

      <service category="tbCreditAGency" name="creditagency"
                                    description="Credit Agency Service">
            <listeners>
                  <jms-listener name="trailblazer-jmscreditagency"
                                    busidref="creditAgencyRequest"
                                    maxThreads="1"/>

            </listeners>
            <actions>
                  <action
            class="org.jboss.soa.esb.actions.converters.SmooksTransformer" name="XMLToCSV">
                    <property name="resource-config" value="/smooks-xml2csv.xml" />
                  </action>
                  <action
            class="org.jboss.soa.esb.samples.trailblazer.actions.CreditAgencyActions"
                        process="processCreditRequest" name="fido">
                  </action>
                  <action
            class="org.jboss.soa.esb.actions.converters.SmooksTransformer" name="CSVToXML">
                    <property name="resource-config" value="/smooks-csv2xml.xml" />
                  </action>
            </actions>
      </service>
      <service category="tbBanks" name="jmsBank" description="JMS Bank Service">
            <listeners>
                  <jms-listener name="trailblazer-jmsbankreq"
                                    busidref="jmsBankRequest"
                                    maxThreads="1"/>

                  </listeners>
            <actions mep="OneWay" >
                  <action class="org.jboss.soa.esb.actions.routing.JMSRouter"
```

```
                                    name="jmsBankAction" >
            <property name="jndiName" value="queue/jms-tb-jmsBankRequestQueue" />
            <property name="unwrap" value="true" />
            </action>
      </actions>
</service>
<service category="tbJmsbank" name="jmsbankreplies"
                        description="Trailblazer JMS Bank Reply Service">
      <listeners>
            <jms-listener name="trailblazer-jmsbankrespgw"
                  busidref="jmsBankResponseGateway"
                  maxThreads="1"
                  is-gateway="true"/>
            <jms-listener name="trailblazer-jmsbankreplies"
                              busidref="jmsBankResponseListener"
                              maxThreads="1"/>
      </listeners>
      <actions>
            <action
            class="org.jboss.soa.esb.samples.trailblazer.actions.BankResponseActions"
                        process="processResponseFromJMSBank" name="pepe"/>
      </actions>
</service>
<service category="tbBanks" name="fileBank" description="File Bank Service">
      <listeners>
            <jms-listener name="trailblazer-filebankreq"
                              busidref="fileBankRequest"
                              maxThreads="1"/>

            </listeners>
      <actions mep="OneWay" >
            <action
      class="org.jboss.soa.esb.samples.trailblazer.actions.FileBankRequestActions"
                               process="processFileBankRequest"
                               name="fileBankRequestAction" >
            </action>
      </actions>
</service>
<service category="tbFilebank" name="filebankreplies"
                              description="Trailblazer File Bank Reply Service">
      <listeners>
            <fs-listener name="FileGateway"
                  busidref="fileBankChannel"
                  maxThreads="1"
                  is-gateway="true"
                  schedule-frequency="10" />
            <jms-listener name="trailblazer-filebankreplies"
                              busidref="fileBankResponseListener"
                              maxThreads="1"/>
      </listeners>
      <actions>
            <action
            class="org.jboss.soa.esb.samples.trailblazer.actions.BankResponseActions"
                        process="processResponseFromFileBank" name="filebankin"/>
      </actions>
</service>
<service category="notifiers" name="customer"
                              description="Customer Notification Service">
      <listeners>
            <jms-listener name="trailblazer-custnotifier"
                              busidref="customerNotifier"
                              maxThreads="1"/>

            </listeners>
      <actions mep="OneWay" >
            <action
      class="org.jboss.soa.esb.samples.trailblazer.actions.NotifyCustomerActions"
                        process="notifyCustomer" name="notify"/>
```

```
                </actions>
            </service>
        </services>

</jbossesb>
```

The config above uses a configuration structure which is described in much more detail in Chapter 5 of the JBossESB Programmer's Guide. The config for the TB describes several communication providers, listed in the *<providers>* section, all consisting of JMS in this example, and using JBossMQ as the actual JMS transport. Next, several *<services>* are listed, starting with the *creditagency*, and the various JMS bank services for sending and receiving data from the banks. The banks have their own config files, which must be configured to use and reply on the queues described above. Please see *<TRAILBLAZER_ROOT>/banks/bank.properties*.

The LoanBroker makes use of the services described above, in the following lines of code:

```
public void processLoanRequest(WebCustomer wCustomer) {

    Customer customer = getCustomer(wCustomer);

    //keep the customer in a file someplace for later use, if needed
    CustomerMasterFile.addCustomer(String.valueOf(customer.ssn), customer);

    //step 1 - send to credit agency for credit score if available

    int score=sendToCreditAgency(customer);

    //added a pause here to give the creditagency some time to reply
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    //step 2 - check if score is acceptable

    if (score >= 4) {
            //step 3a - send to Bank - async
            System.out.println("sending to first Bank...");
            sendToBank(bank1Invoker, customer, "b1");

            System.out.println("sending to second Bank...");
            sendToBank(bank2Invoker, customer, "b2");
    } else {
           //step 3b - notify customer that credit is not acceptable
           String invalidCredit="<insufficientCredit>" +
                        "<customerUID>"+customer.ssn+"</customerUID>" +
                        "<ref>0</ref>"+
                        "<customerEmail>"+customer.email+"</customerEmail>" +
                        "<errorCode>3</errorCode>" +
                        "</insufficientCredit>";

           Message notifyMessage =
                   MessageFactory.getInstance().getMessage(MessageType.JBOSS_XML);
           notifyMessage.getBody().add(invalidCredit);

           try {
                   notifierInvoker.deliverAsync(notifyMessage);
           } catch (Exception e) {
                   logger.error(e);
           }
    }
```

```
    }
```

The sendToCreditAgency is where an interaction with the ESB takes place. Please see the code for more detailed listing. The interaction with the credit agency uses a synchronous invocation, waiting a maximum of 5 seconds for a response.

```
replyMessage = serviceInvoker.deliverSync(message, 5000);
```

After receiving a credit rating score from the CreditAgency service, the loan broker makes a decision (based on whether the score is greater or equal to 4) whether to request quotes from the banks, or to inform the customer their credit is insufficient to proceed.

The interaction with the Banks uses an asynchronous API – there is no waiting for a reply from the banks. The bank replies come in on their own queue, and the GatewayService defined for that purpose fires it off to an action class to handle the response. See the listing from the *jbossesb.xml*:

```
<service category="tbJmsbank" name="jmsbankreplies"
                description="Trailblazer JMS Bank Reply Service">
        <listeners>
                <jms-listener name="trailblazer-jmsbankrespgw"
                        busidref="jmsBankResponseGateway" maxThreads="1"
                                is-gateway="true"/>
                <jms-listener name="trailblazer-jmsbankreplies"
                        busidref="jmsBankResponseListener" maxThreads="1"/>
        </listeners>
        <actions>
                <action
        class="org.jboss.soa.esb.samples.trailblazer.actions.BankResponseActions"
                                process="processResponseFromJMSBank" name="pepe"/>
        </actions>
</service>
```

The important element above is, that the *org.jboss.soa.esb.samples.trailblazer.actions.BankResponseActions* is the class that is defined as being responsible for handling the bank JMS responses. The property *process="processResponseFromJMSBank"* tells the service which method in this class will actually do the work. Below is a code snippet from this method:

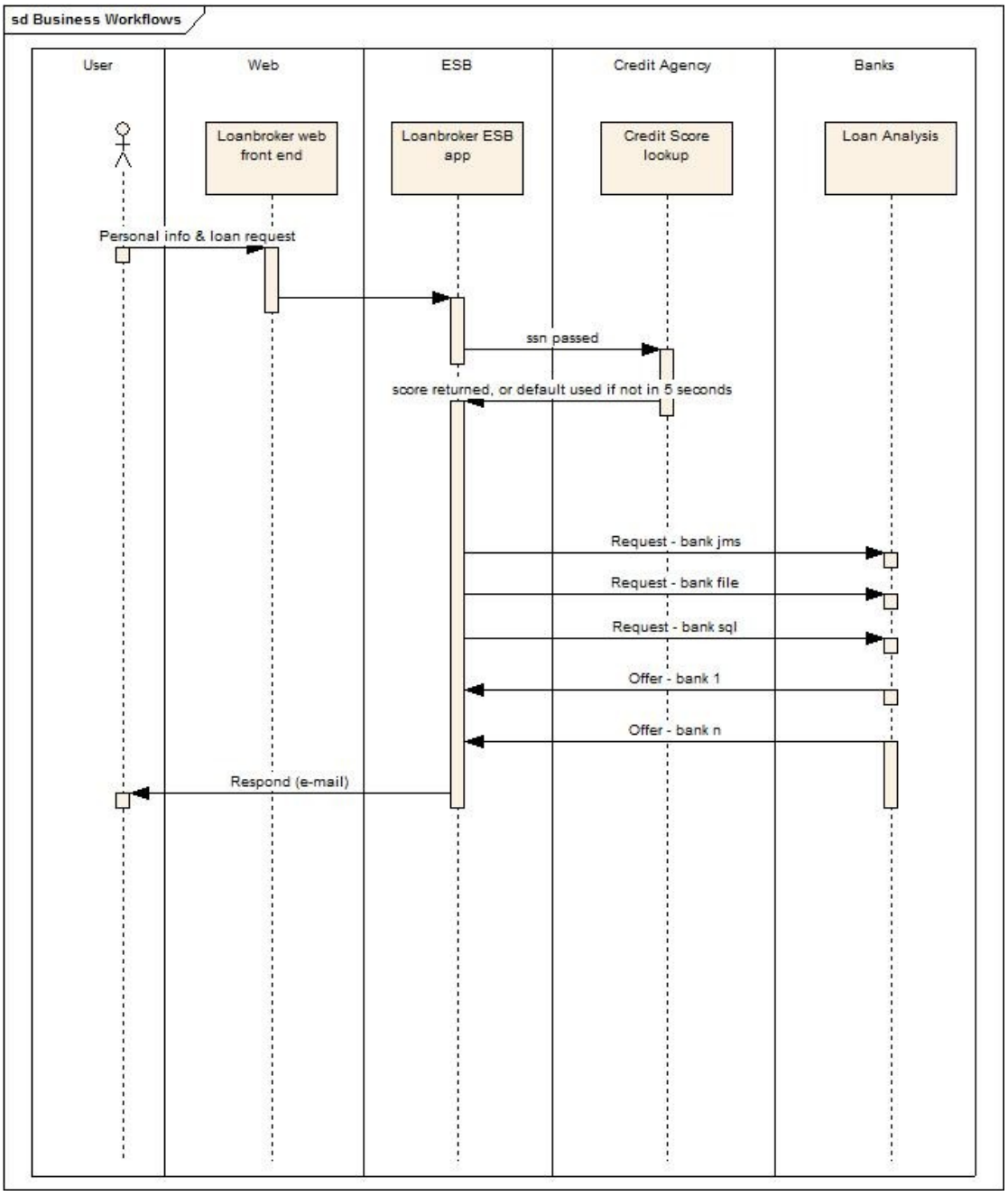
```
public Message processResponseFromJMSBank(Message message) throws Exception {
        System.out.println("Got the message from the JMS bank: " +
                                message.getBody().get());

        _logger.debug("JMS bank message received: \n" + message.getBody().get());

        Message notifyMessage =
                MessageFactory.getInstance().getMessage(MessageType.JBOSS_XML);
        notifyMessage.getBody().add(message.getBody().get());

        notifierInvoker.deliverAsync(notifyMessage);

        return null;
}
```

The code above retrieves the contents of the payload from the Message.getBody().get(). Those contents are then used to construct a new message which is sent to the 'notifier' service (as an asynchronous invocation), which in turn processes the information to fill in the email which goes back to the customer.

The response from the File based Bank is handled in the same way. The only difference is that the *jboss-esb.xml* service configuration uses a different provider (file system as opposed to JMS), and the method called on the action is '*processResponseFromFileBank*'.

The same action class is used for both type of bank responses, and the implementation of the *processResponseFromJMSBank* and *processResponseFromFileBank* actions are identical. Therefore the same 'processResponse' method could be used to handle both responses.

The sequence diagram below illustrates the full set of calls that are made in the trailblazers:



Chapter 3

Deploying and Testing the Trailblazer

System Requirements

1. ANT (1.6.5 or higher)
2. A mail server to send email notifications
3. JBoss ESB Server 4.4GA or JBoss AS 4.2.2.GA with JBossESB installed (see "Getting Started Guide" for installation details).

Configurations

1. **jbossesb-properties.xml**: Update the section titled "transports" and specify all of the SMTP mail server settings for your environment.
2. **deployment.properties**: If you have followed the "Getting Started" guide, you should have already created a *deployment.properties* file using the template in *deployment.properties-example* and set the configuration and directory settings.
3. **trailblazer.properties**: It is located in TB_Root folder, update the file.bank.monitor.directory and file.output.directory properties properly. These are input and output folder need to be specified, they are set to /tmp/input and /tmp/output by default.
4. **jboss-esb.xml**: It is located in the TB_Root/esb/conf folder, there is a "<fs-provider>..</fs-provider>" block, update the "directory" attribute value as same as file.output.directory value in the bank.properties file.

Running

1. Run your JBoss AS – use the *run* script within the *bin* directory. On Windows there is a *run.bat* script, for any Unix variant (Unix/Linux/Mac OS X) there is a *run.sh*.
2. From the TB_ROOT, execute the command to start the ESB: "ant deploy". This should deploy the ESB and WAR files to your JBoss AS server/default.
3. From the TB_ROOT/banks execute the command to start the JMS Bank service: "ant runJMSBank"
4. Open another window/shell, From the TB_ROOT/banks execute the command to start the File Bank Service: "ant runFileBank"
5. In your browser, goto http://localhost:8080/trailblazer and submit some quote requests.
6. Check your email for notifications. You will see either a loan request rejected (single email) because the score is less than 4, or two emails (one from JMS bank and one from FileBased bank) with valid quotes.

Running Trailblazer example with monitor functionality

System Requirements

4. ANT (1.6.5 or higher)
5. A mail server to send email notifications
6. JBoss ESB Server 4.4GA or JBoss AS 4.2.2.GA with JBossESB installed (see "Getting Started Guide" for installation details).

Configurations

1. **jbossesb-properties.xml**: Update the section titled "transports" and specify all of the SMTP mail server settings for your environment.
2. **deployment.properties**: If you have followed the "Getting Started" guide, you should have already created a *deployment.properties* file using the template in *deployment.properties-example* and set the configuration and directory settings.
3. **trailblazer.properties**: It is located in TB_Root folder, update the file.bank.monitor.directory and file.output.directory properties properly. These are input and output folder need to be specified, they are set to /tmp/input and /tmp/output by default.
4. **jboss-esb.xml**: It is located in the TB_Root/esb/conf folder, there is a "<fs-provider>..</fs-provider>" block, update the "directory" attribute value as same as file.output.directory value in the bank.properties file.

Running

1. download the pi4soa tool suite from http://pi4soa.wiki.sourceforge.net/download.
2. create a new Java project and import into Eclipse environment configured with pi4soa tools using the Import->Archive from the Project context menu. The TrailblazerModels.zip should be located in $TB_Root/models folder.
3. In the Eclipse project, from the context menu for the models/TrailBlazer.cdm file.

   select Choreography->Monitor. When it indicates "Monitoring TrailBlazer" on the status line.
4. from the TB_ROOT, execute the command to deploy monitor related packages: "ant deploy-monitor-lib".
5. Following Running instructions in the "Deploying and Testing the Trailblazer" above.
6. Repeat the loan request step multiple times, remembering to change the SSN number to create new transactions in the monitor.
7. once you finish your testing, you can run command "ant undeploy-monitor-lib" to remove the monitor library and filter.