# Loan Broker Tutorial

The Open Source SOA Platform

09/06 Version 1.2.1

## Loan Broker Tutorial

One of the main reasons for having a service oriented architecture is to integrate incompatible existing applications. The Loan Broker demonstration is a good example of an application requiring a service oriented architecture. The Loan Broker application is composed of three banks and a credit agency all modeled on different protocols. The demo leverages JBI features and uses a BPEL engine and a WS-Notification broker to show the integration of several otherwise unrelated applications.

# Contents

# 1. Overview

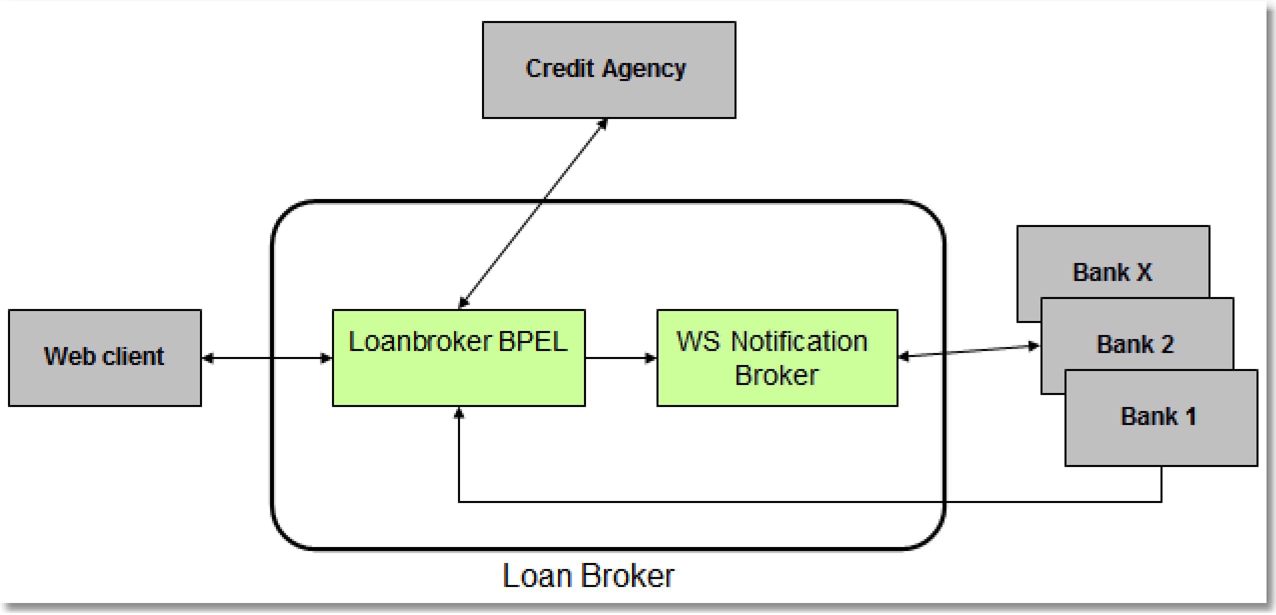


*Figure 1: Loan Broker System*

The Loan Broker is patterned after the `ComposedMessagingTIBCO.html` pattern from the *Enterprise Integration Patterns* book. Quoting from *Enterprise Integration Patterns*, "Our application is a simple bank quote request system. Customers submit quote requests to a loan broker interface. The loan broker fulfills the request by first obtaining a credit score, then requesting quotes from a number of banks. Once the Loan Broker obtains quotes from the banks it selects the best quote and returns it to the customer".

The diagram shows:

1. The web client requests a loan rate from the Loan Broker system.
2. The loan broker BPEL service engine asks the Credit Agency for the credit rating of the loan requester.
3. The loan broker solicits the best loan rate from multiple banks. The WS-Notification service engine sends the request loan amount, duration, and the loan requester's credit score to the banks.
4. The banks respond to the loan broker with their best loan rate.
5. The loan broker determines which bank had the best rate and returns that information to the requester (web client).

## 2. Running the Demo

The loan broker demo is pre-installed and ready to run as part of the LogicBlaze FUSE distribution. To run the demo follow the instructions below.

If you would like to try deploying the service assemblies from scratch follow the instructions in the *Installing from Scratch* section of this tutorial.

To run the demo:

1. In a browser, go to `http://localhost:8080/loanbroker-client/`

   Using the Mozilla Firefox browser is recommended.

2. Enter a nine digit number in the SSN field. This can be entered with or without the dashes, but must start with a "1". For example, 199-99-9999 or 199999999.
3. Enter a number in the Loan Amount field, such as, 50000.00.
4. Enter a number in the Duration field, such as 10. It is assumed to be a number of years.
5. Click the Send button. The best loan rate will be displayed back to your browser.
6. Alternatively, you can send 10 random loan rate requests by clicking the Send 10 random requests button.

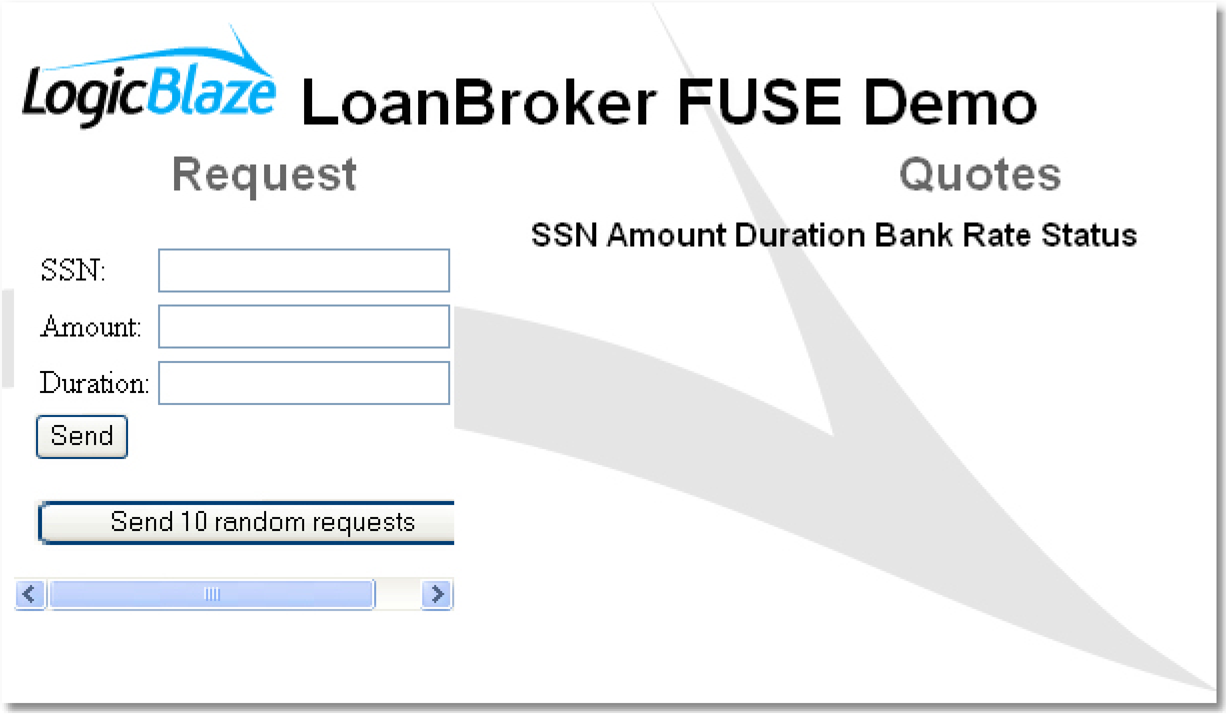
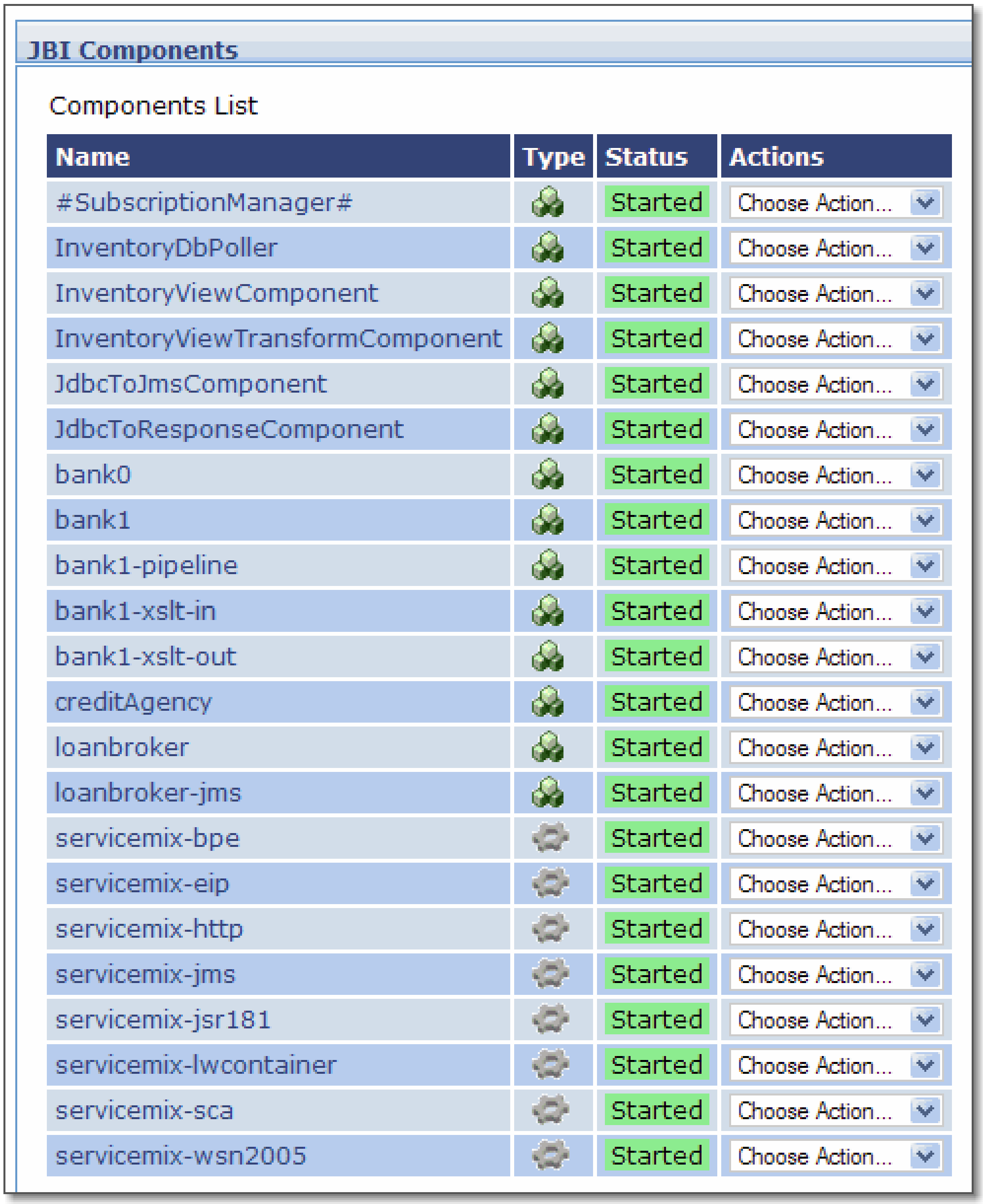


*Figure 2: Loan Broker Client*

Use the LogicBlaze FUSE Console to see all the deployed service assemblies and to try starting and stopping them.

1. Open a browser and navigate to the FUSE Console.
2. Scroll down to the JBI Components portlet. You can stop any of the loan broker components, for example, `creditAgency`. Use the pull-down menu under "Actions" and select Stop.



**JBI Components**

Components List

| Name | Type | Status | Actions |
|---|---|---|---|
| #SubscriptionManager# | ♣ | Started | Choose Action... ▾ |
| InventoryDbPoller | ♣ | Started | Choose Action... ▾ |
| InventoryViewComponent | ♣ | Started | Choose Action... ▾ |
| InventoryViewTransformComponent | ♣ | Started | Choose Action... ▾ |
| JdbcToJmsComponent | ♣ | Started | Choose Action... ▾ |
| JdbcToResponseComponent | ♣ | Started | Choose Action... ▾ |
| bank0 | ♣ | Started | Choose Action... ▾ |
| bank1 | ♣ | Started | Choose Action... ▾ |
| bank1-pipeline | ♣ | Started | Choose Action... ▾ |
| bank1-xslt-in | ♣ | Started | Choose Action... ▾ |
| bank1-xslt-out | ♣ | Started | Choose Action... ▾ |
| creditAgency | ♣ | Started | Choose Action... ▾ |
| loanbroker | ♣ | Started | Choose Action... ▾ |
| loanbroker-jms | ♣ | Started | Choose Action... ▾ |
| servicemix-bpe | ⚙ | Started | Choose Action... ▾ |
| servicemix-eip | ⚙ | Started | Choose Action... ▾ |
| servicemix-http | ⚙ | Started | Choose Action... ▾ |
| servicemix-jms | ⚙ | Started | Choose Action... ▾ |
| servicemix-jsr181 | ⚙ | Started | Choose Action... ▾ |
| servicemix-lwcontainer | ⚙ | Started | Choose Action... ▾ |
| servicemix-sca | ⚙ | Started | Choose Action... ▾ |
| servicemix-wsn2005 | ⚙ | Started | Choose Action... ▾ |

*Figure 3: Deployed Components*

# 3. Installing from Scratch

It is instructive to see how to install the demo from scratch. You can install/deploy manually from a Windows command window or Unix shell, or you can install/deploy using the LogicBlaze FUSE Console.

## 3.1. Installing and Starting the Demo from a Command Window

This procedure shows how to install and start the loan broker demo from a command shell. The procedure uses Windows style commands and the Windows specific backslash. However, the same steps apply for Unix systems, with the appropriate substitutions of "*cp*" for "*copy*" and forward slash replacing the backslashes. Of course you may use operating system tools, such as Windows Explorer, as well.

> LogicBlaze FUSE does not need to be running to deploy the demo. The service assembly jar files could be copied to the `hotdeploy` directory prior to bringing up LogicBlaze FUSE. Upon start-up the demo would deploy. The following steps show how to hot deploy service assemblies.

1. By default the demo is preinstalled, therefore, you must uninstall it. Before uninstalling, put a copy of the service assembly jar files in a safe location.

```
cd [fuse_dir]\hotdeploy
copy loanbroker-assembly-1.0.jar \tmp
copy creditagency-assembly-1.0.jar \tmp
copy bank0-assembly-1.0.jar \tmp
copy bank1-external-assembly-1.0.jar \tmp
copy bank1-internal-assembly-1.0.jar \tmp
```

where `[fuse_dir]` is the directory in which LogicBlaze FUSE was installed.

2. Undeploy the service assemblies (see the *Undeploying* section of this document), either manually or using the LogicBlaze Console. Then proceed to step three of this procedure.
3. LogicBlaze FUSE must be running. For instructions to start LogicBlaze FUSE, please see the *Getting Started with LogicBlaze FUSE* guide.

> Please allow time for LogicBlaze FUSE to completely start-up.

4. From a command window or shell window install the service assemblies for the loan broker application. The directory pathnames are different for a binary distribution and a source distribution.

   For a binary distribution:

   `cd \tmp` - if this is where you saved the JAR files, otherwise use your directory.

   ```
   copy loanbroker-assembly-1.0.jar [fuse_dir]\hotdeploy
   copy creditagency-assembly-1.0.jar [fuse_dir]\hotdeploy
   copy bank0-assembly-1.0.jar [fuse_dir]\hotdeploy
   ```

   where `[fuse_dir]` is the directory in which LogicBlaze FUSE was installed.

   For a source distribution:

   `cd \tmp` - if this is where you saved the JAR files, otherwise use your directory.

   ```
   copy loanbroker-assembly-1.0.jar
      [fuse_dir]\target\fuse-1.x\fuse-1.x\hotdeploy
   copy creditagency-assembly-1.0.jar
      [fuse_dir]\target\fuse-1.x\fuse-1.x\hotdeploy
   copy bank0-assembly-1.0.jar
      [fuse_dir]\target\fuse-1.x\fuse-1.x\hotdeploy
   ```

   where `[fuse_dir]` is the directory in which LogicBlaze FUSE was installed.

   The above steps must be done in a different command window from the one in which LogicBlaze FUSE was started.

5. Allow time for the service assemblies (SAs) to be deployed. To see them deploy, watch the command window where LogicBlaze FUSE was started.
6. You may have noticed that only one bank (bank 0) was deployed above. To demonstrate that the banks can be deployed separately you can deploy the other banks now or you can run the demo. An interesting point is that components, service units, and service assemblies can be "hot" deployed after LogicBlaze FUSE is already running. To deploy the other banks:

   `cd [fuse_dir]\tmp` - if this is where you saved the JAR files, otherwise use your directory.

   ```
   copy bank1-external-assembly-1.0.jar [fuse_dir]\hotdeploy
   copy bank1-internal-assembly-1.0.jar [fuse_dir]\hotdeploy
   ```

7. You are now ready to run the demo from a Web browser. Proceed to the Running the Demo section for instructions.

## 3.2. Installing and Starting the Demo from the LogicBlaze FUSE Console

To install and start from the console:

1. By default the demo is preinstalled, therefore, you must uninstall it. Before uninstalling, put a copy of the service assembly jar files in a safe location.

   ```
   cd [fuse_dir]\hotdeploy

   copy loanbroker-assembly-1.0.jar \tmp
   copy creditagency-assembly-1.0.jar \tmp
   copy bank0-assembly-1.0.jar \tmp
   copy bank1-external-assembly-1.0.jar \tmp
   copy bank1-internal-assembly-1.0.jar \tmp
   ```

   where `[fuse_dir]` is the directory in which LogicBlaze FUSE was installed.
2. Go to the *Undeploying* section of this document and undeploy the service assemblies, either manually or using the LogicBlaze Console. Then proceed to step three of this procedure.
3. LogicBlaze FUSE must be running. For instructions to start LogicBlaze FUSE, please see the *Getting Started with LogicBlaze FUSE* guide.

   Please allow time for LogicBlaze FUSE to completely start-up.
4. Open the LogicBlaze FUSE Console. In a browser go to: `http://localhost:8080/`, and navigate to access the LogicBlaze FUSE console. For more information on the console please visit the *LogicBlaze FUSE Console Guide*.

   `localhost` should be replaced with the `hostname` where LogicBlaze FUSE is installed. If port 8080 has been reassigned you must change that as well.

5. The demo can be deployed from the Archives Installed/Deployed portlet. You may have to scroll down in your browser to see this portlet. Click on Browse to select the service assembly to deploy. Then click on the Install/Deploy button.
6. Deploy the following service assemblies:

   ```
   loanbroker-assembly-1.0.jar
   creditagency-assembly-1.0.jar
   bank0-assembly-1.0.jar
   bank1-external-assembly-1.0.jar
   bank1-internal-assembly-1.0.jar
   ```

   You saved these to another directory in an earlier step, e.g., `/tmp`

7. When the service assemblies are deployed they will be displayed in the JBI Components portlet and Archives Installed/Deployed portlet in the console. By default they are deployed in a Started state.

8. You are now ready to run the demo from a Web browser. Proceed to the *Running the Demo* section for instructions.

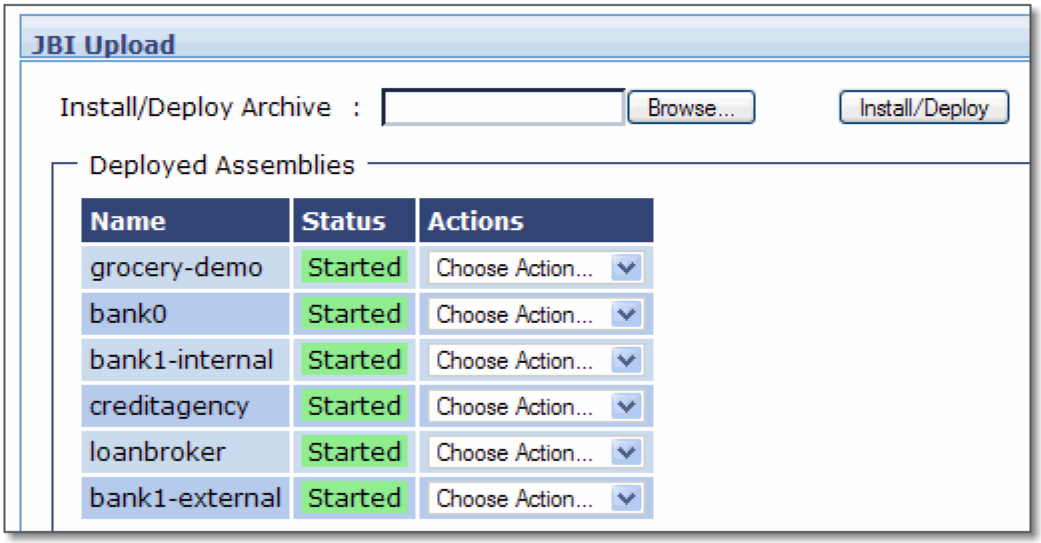
# 4. Undeploying

Components and service assemblies can be undeployed manually or using the LogicBlaze FUSE Console.

This is the list of service assemblies that you must uninstall to fully undeploy the loan broker demo.

```
loanbroker-assembly-1.0.jar
creditagency-assembly-1.0.jar
bank0-assembly-1.0.jar
bank1-internal-assembly-1.0.jar
bank1-external-assembly-1.0.jar
```

## 4.1. Undeploying from the LogicBlaze FUSE Console



| Name | Status | Actions |
|---|---|---|
| grocery-demo | Started | Choose Action... |
| bank0 | Started | Choose Action... |
| bank1-internal | Started | Choose Action... |
| creditagency | Started | Choose Action... |
| loanbroker | Started | Choose Action... |
| bank1-external | Started | Choose Action... |

*Figure 4: Archives Installed/Deployed Portlet*

1. LogicBlaze FUSE must be running. For instructions to start LogicBlaze FUSE, please see the Getting Started with LogicBlaze FUSE guide.
2. In a browser go to: `http://localhost:8080/` and navigate to access the LogicBlazeFUSE console. For more information on the console please visit the *LogicBlaze FUSE Console Guide*.
3. In the Archives Installed/Deployed portlet, select Stop from the "Actions" pull-down menu next to the component you are undeploying.

   This puts the component into the Stopped state. You can either leave the component installed and stopped or you can proceed to the next step to continue with undeploying the component.
4. In the Archives Installed/Deployed portlet, select Shutdown from the "Actions" pull-down menu for the component you want to undeploy. The component is still installed (deployed), but in the Shutdown state. To uninstall/undeploy it proceed to the next step.

5. In the Archives Installed/Deployed portlet, select the Uninstall from the "Actions" pull-down menu for the component you want to undeploy. That's it, the component is undeployed!
6. Repeat steps 3 - 5 for each component you want to uninstall.

## 4.2. Manual Undeployment

To undeploy the components or service assemblies manually:

1. Remove the components from the [fuse_dir]\hotdeploy directory by deleting the jar files.

   This step can be done when LogicBlaze FUSE is running or stopped.

2. If LogicBlaze FUSE was not running, start it. Please see the Getting Started with LogicBlaze FUSE guide for instructions if needed .

After start-up if the components still seem to be deploying:

1. Shutdown LogicBlaze FUSE.
2. Delete the `[fuse_dir]\data` directory.

   **Warning**: This is OK on a test system, but should not be done on a production installation of LogicBlaze FUSE because information about all other deployed components will be lost.

3. Restart LogicBlaze FUSE.

# 5. How the Loan Broker Demo Works

The following sections provide more technical details about how this demo works and how it was packaged.

## 5.1. Packaging

The demo is comprised of several service assemblies. Recall that a service assembly (SA) is a standard way to package services together as a composite service/application to be deployed as a unit.

**loanbroker-assembly**

This service assembly contains the BPEL process and related bindings:

- `loanbroker-bpel` - the asynchronous `bpel` process
- `loanbroker-lb` - synchronous front-end to the `bpel` process
- `loanbroker-ca` - http binding to the `CreditAgency`
- `loanbroker-jms` - JMS binding (provider) for the synchronous front-end

**creditagency-assembly**

This service assembly represents the external `creditagency` service. In the real world, this service may not be implemented using JBI.

- `creditagency-service` - the `creditagency` service implementation
- `creditagency-http` - http binding (consumer) for the above service

**bank0-assembly:**

This service assembly contains a very simple bank implementation.

- `bank0-wsn` - WS-Notification subscription
- `bank0-service` - bank service implementation

**bank1-internal-assembly:**

This service assembly is the consumer side for Bank1.

- `bank1-internal-wsn` - WS-Notification subscription
- `bank1-internal-xslt` - XSLT transformations
- `bank1-internal-http` - http binding (provider) for the external Bank1 service.

**bank1-external-assembly**

This service assembly represents the external Bank1 service. In the real world, this service may not be implemented using JBI.

- `bank1-external-service` - simple Bank1 implementation
- `bank1-external-http` - http binding (consumer) for the above service

## 5.2. Message Flow

First let's see how a single loan request is processed by following a loan rate request from when the request is made until a loan rate result is returned. The diagram shows the flow of the loan rate request message.

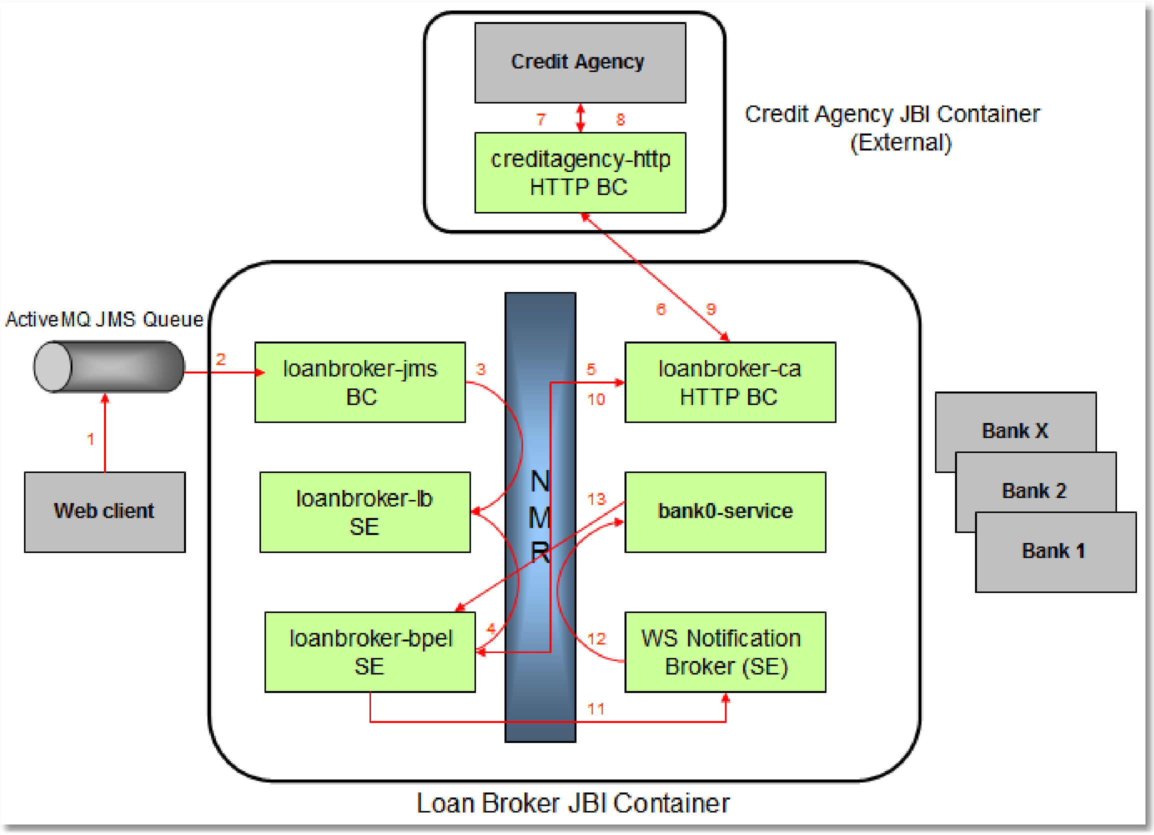


*Figure 5: Message Flow*

The demo is implemented with two banks. In a real world situation, the Credit Agency and the Banks would most likely be external to the loan broker application - in fact, they would typically be external companies. To keep the demo simple, Bank 0 has been implemented as a service internal to the Loan Broker JBI Container. Bank 1 - Bank X are shown in the diagram as external entities. They would each connect to the loan broker JBI system via a binding component. The binding components for the various banks would communicate in a protocol the particular bank was using.

The diagram shows the message flow for one loan rate request:

1. The Web client is the loan requester It requests a loan rate by publishing a JMS message on an Apache ActiveMQ queue.

2. `loanbroker-jms` receives the message. `loanbroker-jms` is a binding component (BC) that can receive and process JMS messages. `loanbroker-jms` will normalize the incoming message. Normalizing the message will put it in the neutral format that is used within the JBI system. After normalization, `loanbroker-jms` will pass the message along to the intended recipient via the Normalized Message Router (NMR). In this case the recipient is `loanbroker-lb`. Note that `loanbroker-jms` implements the In-Only Message Exchange Pattern (MEP). Please see JSR 208 and Introduction to Apache ServiceMix for details about message exchange patterns.
3. `loanbroker-lb` is a front-end to the BPEL process. `loanbroker-lb` is a service engine (SE) and its job is to take the loan request and pass it to the `loanbroker-bpel` process. It then waits for a reply from the BPEL process. The `loanbroker-lb` is an aggregator; it receives an In-Out MEP , sends an In-Only MEP to the `loanbroker-bpel`, and waits for an incoming In-Only MEP on a callback endpoint. This response is sent back to the consumer as the Out message of the In-Out MEP.
4. The `loanbroker-bpel` process receives the loan rate request. As the BPEL process is asynchronous, the incoming MEP for `loanbroker-bpel` is In-Only and the response is sent as an In-Only MEP to a callback endpoint.
5. First `loanbroker-bpel` must obtain the credit rating of the requester. To do this the BPEL process sends a message over the NMR to the `loanbroker-ca` process.
6. `loanbroker-ca` is a binding component. It translates the normalized message (which contains a credit rating request) into an HTTP+SOAP message. This is done so `loanbroker-ca` can communicate with the `creditagency-http` binding component. The MEP for `loanbroker-ca` is In-Out.
7. The `creditagency-http` BC passes the credit rating request to the `creditagency-service`.

In our actual implementation, the `creditagency-service` is implemented inside of the JBI bus as a service engine. However, the diagram shows it outside as a separate JBI container.. In a real world situation the credit agency would most likely be an external entity, probably at a another company or at least on a different server. For simplification of the demo code it is implemented as another service engine internal to the Loan Broker JBI container, but the diagram is showing the more likely scenario.

8. The `creditagency-service` processes the request and responds. The response is sent back via a SOAP message over HTTP to the `creditagency-http` binding component.
9. The `creditagency-http` binding component receives the HTTP+SOAP response from the `creditagency-service` and passes it to the `loanbroker-ca` binding component, also via HTTP+SOAP.
10. `loanbroker-ca` normalizes the HTTP+SOAP message and sends it over the NMR to the `loanbroker-bpel` process.

11. Now that the BPEL process has the credit score information, it can send the loan rate request to the banks. The loan request for the banks is sent to a WS-N process. This component would normally notify a group of banks of the request. The message exchange pattern for `loanbroker-bpel` is In-Only.

    ---

    In our example there are only two banks implemented, but many more could be added. Furthermore, Bank 0 has been implemented internal to the JBI container as a service engine component. Typically, the bank would be external to the JBI system.

    ---

12. `bank0-wsn` sends the loan request to Bank 0. The MEP is In-Only.
13. `bank0-service` processes the loan request and sends a response back to `loanbroker-bpel` using an In-Only MEP.

    The `BPEL` process and the `WS-N` process loop. The `BPEL` process sends the loan rate request to the to the banks via `WS-N`. When the `BPEL` process receives a reply from a bank it compares the rate to the last best rate it had. If the new rate is better, the `BPEL` process sends another message to all the banks via `WS-N` giving them the opportunity to beat the rate. This means that the `WS-N` process is also looping because when it gets another message from `BPEL` with the latest best rate it, in turn, sends out another request to all the banks. When a given amount of time has elapsed and no bank has made a better offer, the last best quote is returned to the requester (Web client).

In an attempt to simplify this explanation, the message flow for bank 1 - bank X is not shown. The scenario outlined above is generally the same for the external banks. Except they connect into the Loan Broker system via binding components because they are not internal to the JBI as Bank 0 is.

## 5.3. Enterprise Integration Pattern

Another way to look at what is occurring in the loan broker system is to look at its enterprise integration pattern.

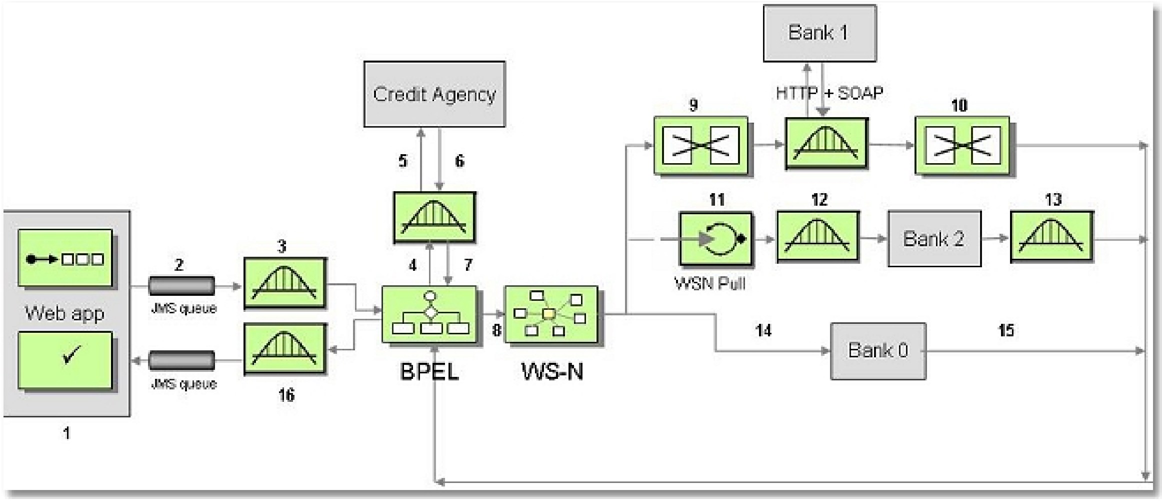


*Figure 6: Loan Broker Enterprise Integration Pattern*

The diagram graphics are from Enterprise Integration Patterns. For a list of what each graphic symbolizes see: http://www.enterpriseintegrationpatterns.com/toc.html.

In general all the green boxes represent components. The green boxes with the bridge symbol are binding components. These components will normalize incoming messages. All other green boxes represent service engine components. They perform some processing on the messages. All the green boxes are within the Loan Broker JBI system and communicate with each other over the NMR. The gray boxes represent external services, such as the Banks and the Credit Agency.

Explaining the diagram from left to right:

1. The leftmost gray box (labeled "1") represents the web client. It sends a message to the loan broker system to request a loan rate. Note that the green boxes symbolize sending a test message.
2. The loan rate request message is sent via a JMS queue to a message bridge.
3. The messaging bridge connects two messaging systems to each other. In other words, the JMS queue is connected to the loan broker via the message bridge. In the JBI environment the message bridge is a binding component that normalizes the message.
4. A process manager routes messages through a series of dynamic steps. This "process manager" is BPEL. In the loan broker application, BPEL obtains credit scores from the Credit Agency and it orchestrates sending the loan rate request to the banks, which will be seen in a later step. In step 4, BPEL sends a message over a bridge to the Credit Agency.
5. The Credit Agency is external to the Loan Broker system. The loan broker messaging system must be bridged to the messaging system of the Credit Agency. The bridge will translate the message to a protocol understood by the Credit Agency.
6. The Credit Agency returns the credit score to the loan broker, via the message bridge.
7. The loan broker (BPEL process) now has enough information about the requestor to send the the loan request to several banks. It passes the loan request to the WS-Notification (WS-N) process.
8. WS-N is a standard based Web services approach to notification using a topic-based publish/subscribe pattern. In this case, WS-N is acting as a message broker between the loan broker and the banks. WS-N will publish the loan request to an internal JMS topic and the interested banks will subscribe to the topic to receive the loan request.
9. Bank 1 cannot process the loan request because it is in an incompatible format. Therefore, the message is first processed by a message translator. The translator puts the message into the format Bank 1 can accept, in this case SOAP over HTTP.
10. After Bank 1 processes the request it returns a response to the BPEL process manager. First it translates it back into the original message format of a JMS message.
11. In the meantime, Bank 2 uses a polling method to receive the loan request. Polling consumers are used when a receiver wants to control when it receives messages. This is described in Enterprise Integration Patterns, "This also known as a synchronous receiver, because the receiver thread blocks until a message is received. We call it a Polling Consumer because the receiver polls for a message, processes it, then polls for another."
12. After processing the request, Bank 2 returns a response message to the BPEL process manager.

13. Bank 2's response passes through the bridge to get translated back to a normalized format which the loan broker system understands.
14. In the meantime, Bank 0 accepts the loan rate request message and processes. Bank 0 is a special case because it is actually internal to the loan broker system. It is in the same JBI container as the rest of the loan broker components. Therefore, it can accept the message in the normalized format.
15. Bank 0 sends a response to the BPEL process.
16. The BPEL process has been looping for the duration of this loan request. As the banks respond, BPEL keeps track of the best rate seen so far. After each response and comparison, BPEL sends out another request giving all the banks a chance to offer a better rate. At that time a timer is started. If the timer times out, then BPEL knows it has the best rate and returns that to the requester.

# 6. Additional Resources

For information on MEPs, binding components, service engine components, NMR, etc., please see:

*Introduction to Apache ServiceMix*

Apache ServiceMix Wiki: http://incubator.apache.org/servicemix/

To read the Java Business Integration specification (JSR 208) please see: http://www.jcp.org/en/jsr/detail?id=208.

For online information about the Enterprise Integration Patterns book, please see: http://www.enterpriseintegrationpatterns.com/ComposedMessagingTIBCO.html.

For online information about the Enterprise Integration Patterns graphics, please see: http://www.enterpriseintegrationpatterns.com/toc.html.