Grocery Inventory Tutorial

09/06 Version 1.2.1

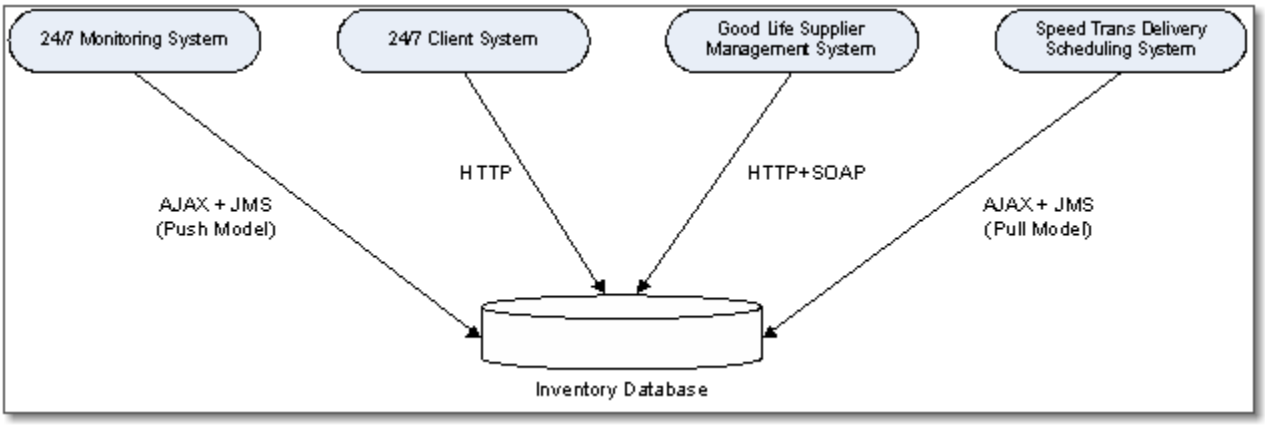
Grocery Inventory Tutorial

The grocery demo aims to exhibit the connectivity of a SOA enabled enterprise application system by using the FUSE SOA platform.

Contents

1. Overview of the Grocery Inventory Demonstration

The Grocery Inventory Demonstration is a simulation of an inventory system for a grocery store. You can view the inventory of the store directly, let a supplier check the stock of an item they inventory, let a partner monitor inventory, and send transactions (random or user specified) to change inventory.



The demo consists of a single enterprise application by the company 24/7 grocery store. It consists of a main inventory database that is accessed over different binding protocols and exposed to different services.

Business Entities:

- *Inventory Database* – the inventory database contains the data owned by the 24/7 grocery store and is queried by its own applications and external business partner applications.
- *24/7 Monitoring System* – AJAX based system that continually displays the quantities of the items in the inventory system of the 24/7 grocery store.
- *24/7 Client System* – simple HTTP based system that simulates a client application that updates the inventory database (e.g., POS system).
- *Good Life Supplier Management System* – simulates an external system that accesses the inventory database via a Web service. In this case, the client is a supplier for the 24/7 grocery store that wants to check the level of the items it is supplying by querying the database of the store.
- *Speed Trans Delivery Scheduling System* – simulates an external system that accesses the inventory database via a JMS client. Speed Trans publishes its request and waits for a reply. In this case, Speed Trans is a partner company that delivers different products from different grocery companies. It queries its partner groceries for availability of specific items.

The demo was implemented using two approaches : (1) the **Ajax Demo** shows the use of Ajax and a ServiceMix JDBC component, and (2) the **Non-Ajax Demo** shows the use of a `servlet/jsp` front-end interface. Both approaches were implemented to illustrate the differences between them. As you will see if you run both of them, the Ajax Demo is a much more efficient implementation.

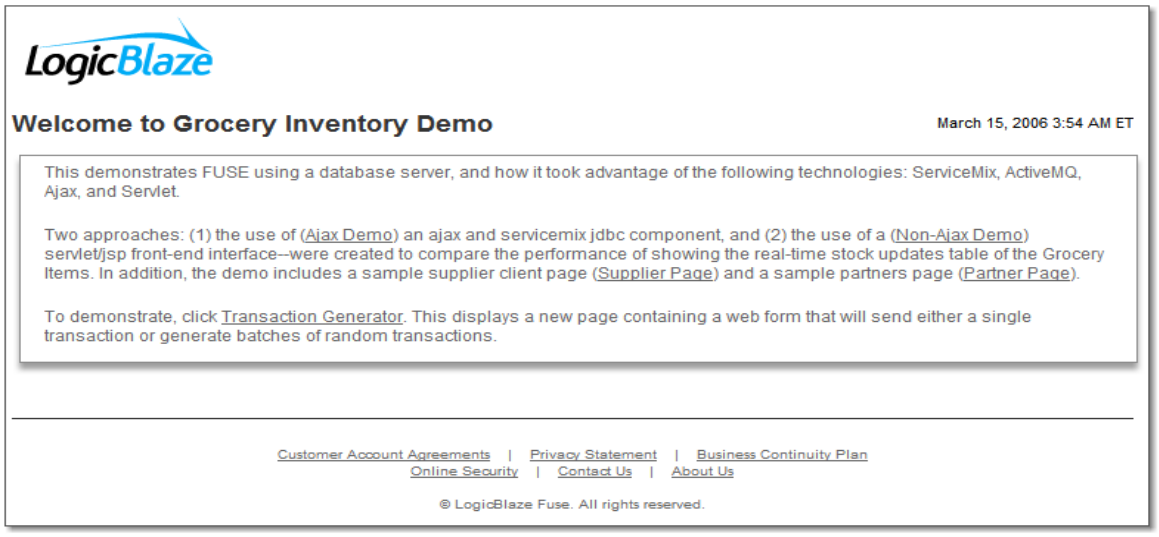
2. Ajax Implementation

This section shows the details of the first (and preferred) approach using Ajax.

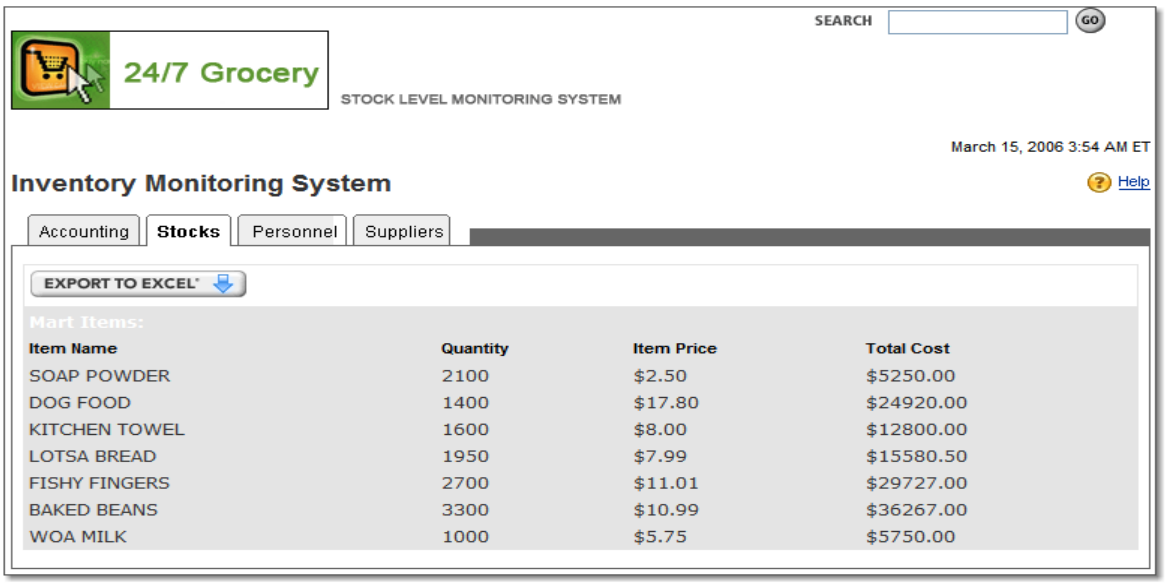
2.1. Running the Demonstration

To run the Ajax demo perform the following steps:

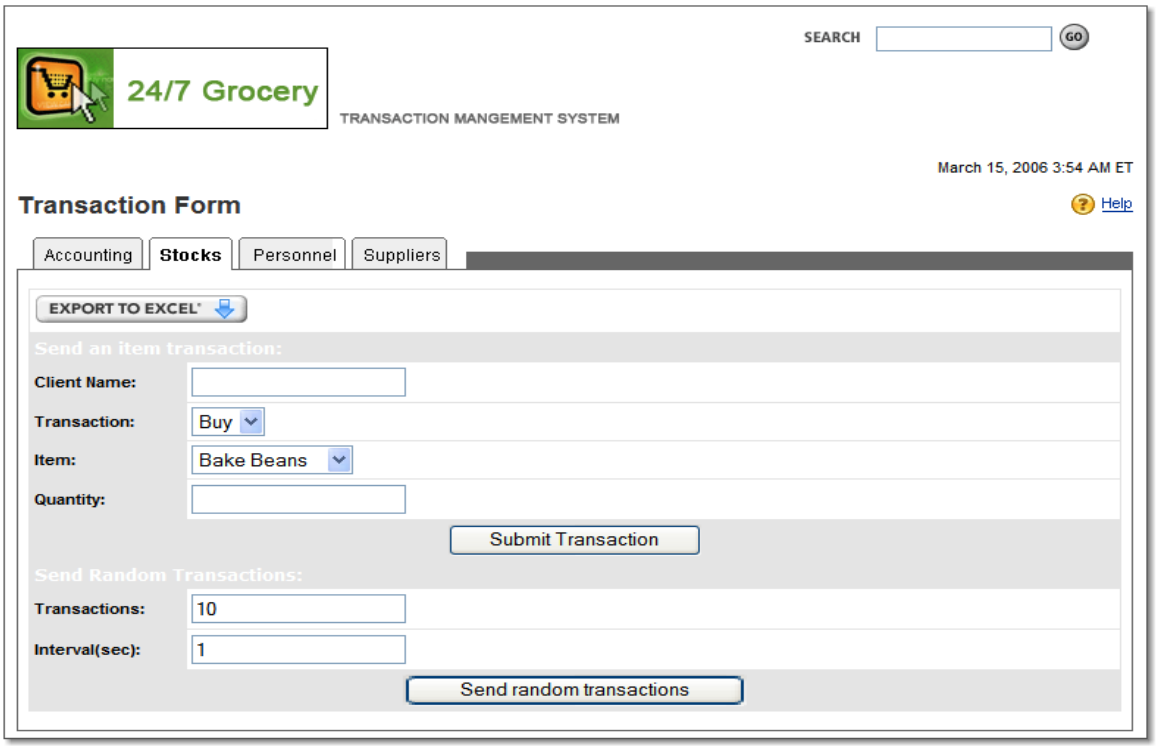
1. Go to http://localhost:8080/grocery-demo/



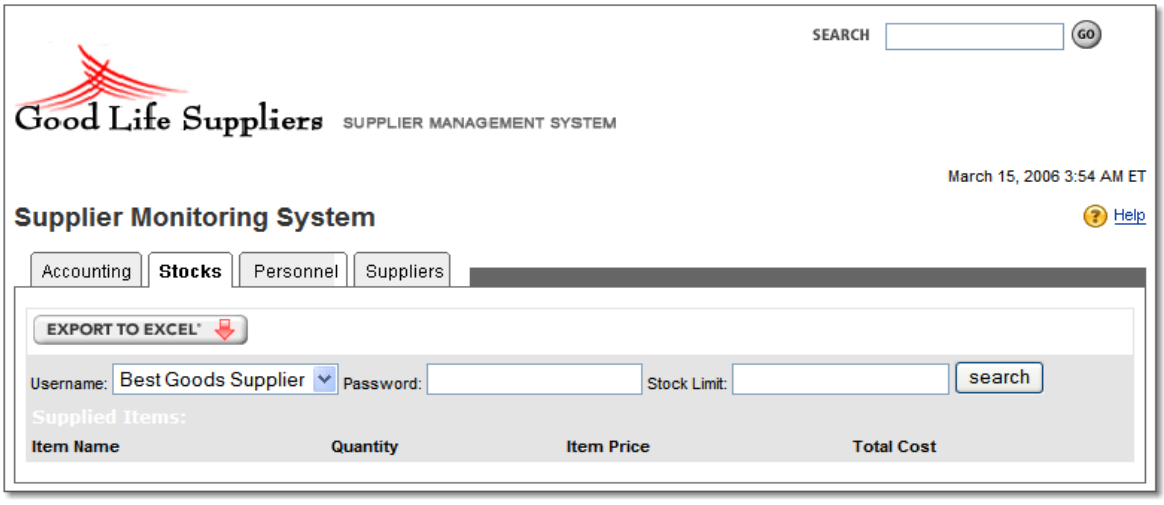
2. Click on the “Ajax Demo” link. This will show you the inventory of the “24/7 Grocery” store.



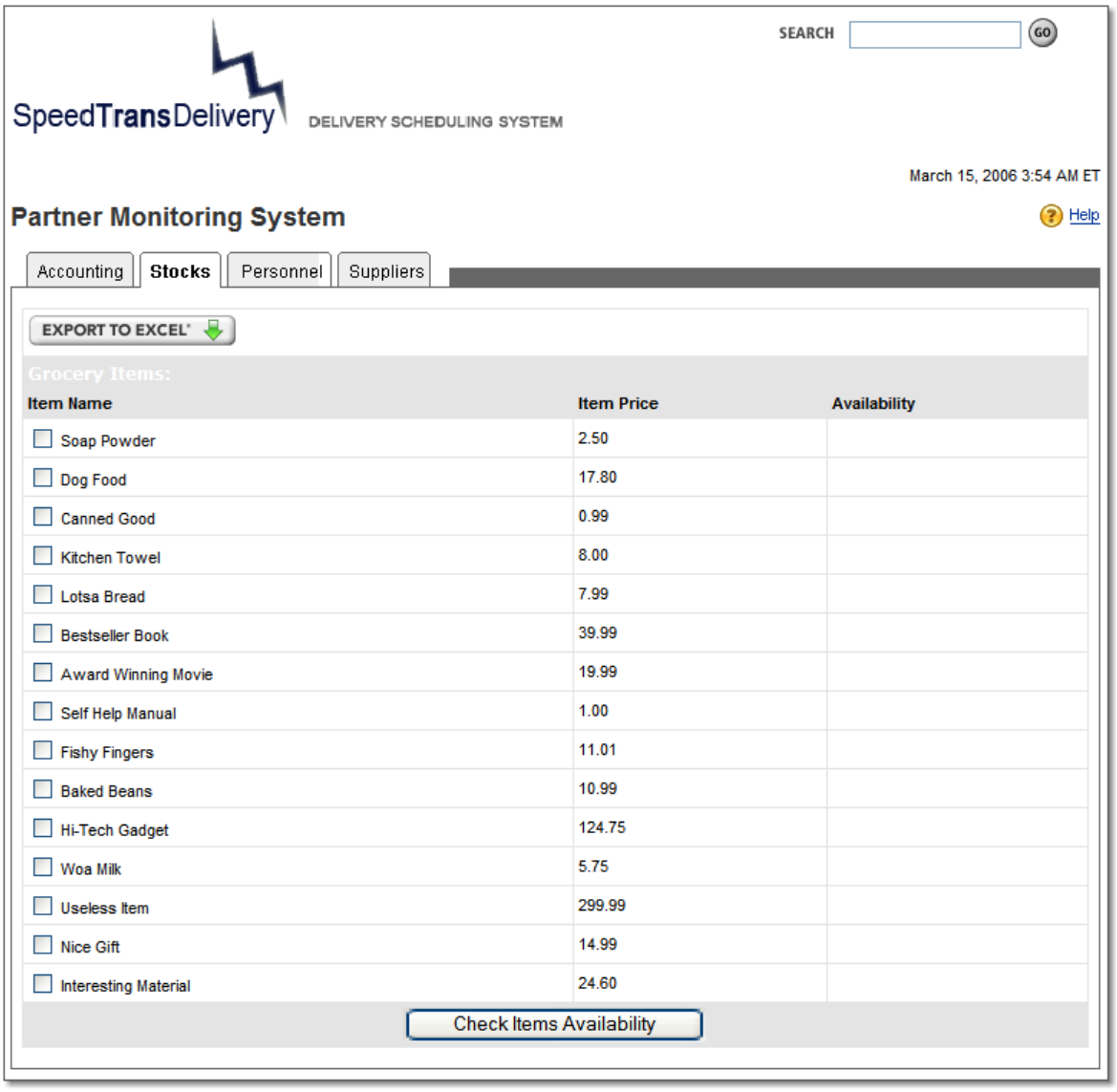
3. Click the link to open the “Transaction Generator”. You may chose to specify a value and submit a single transaction or send 10 random transactions. Please watch your “Inventory Monitoring System” window to see the results of this action.



4. Click the link to open the “Supplier Page”. Enter a value in the `Stock Limit` field. Click “Search” to see what items have gone below the limit. The “Password” required to run this portion of the demo is “password”.



5. Click the link to open the “Partner Page”. Select as many check boxes as you wish in “Grocery Items” column and click “Check Items Availability” to view the availability of each individual grocery item.



2.2. How the Grocery Inventory Demo Works

The diagrams below show the architecture we have used to implement this FUSE + Ajax application. The first diagram is a complete view of the demonstration. For simplicity the main diagram has been broken into many diagrams (shown later) representing each of the individual systems.

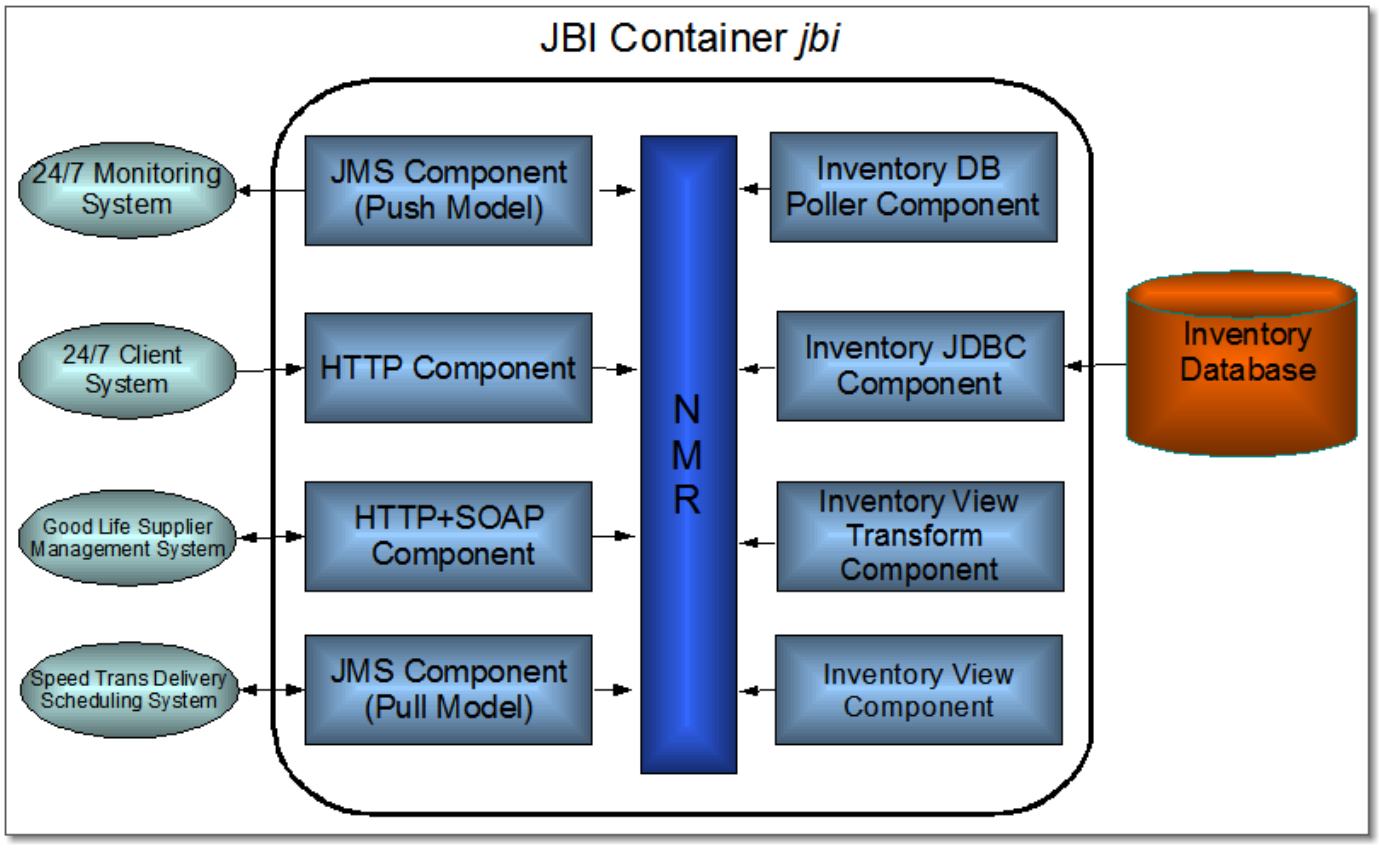


Figure 1: All Systems View

The grocery demo consists of JBI services, binding components, and actual client applications that access a single inventory database.

1. *HTTP Component* – a HTTP binding component that gives HTTP clients access to the inventory database.
2. *HTTP + SOAP Component* – SOAP enabled HTTP binding component that exposes a Web service that provides access to the inventory database.
3. *JMS (Push Model) Component* – JMS binding component that allows a JBI service to publish JMS messages to an external topic.
4. *JMS (Pull Model) Component*– JMS binding component that provides a request reply approach to querying the database.

5. *Inventory DB Poller Component* – JBI service that constantly sends an update request to the inventory JDBC `component` that tells it to publish a list of updated items, since the last check.
6. *Inventory View Component* – JBI service that chains together the `Inventory View Transform Component` and `Inventory JDBC Component` to provide a limited access to the Inventory DB with simple authentication mechanism and query checking.
7. *Inventory View Transform Component* – JBI service that filters a request to the inventory JDBC `component` that limits the query that can be perform on the database.
8. *Inventory JDBC Component* – JBI service that provides direct access to the inventory database.

24/7 Client System

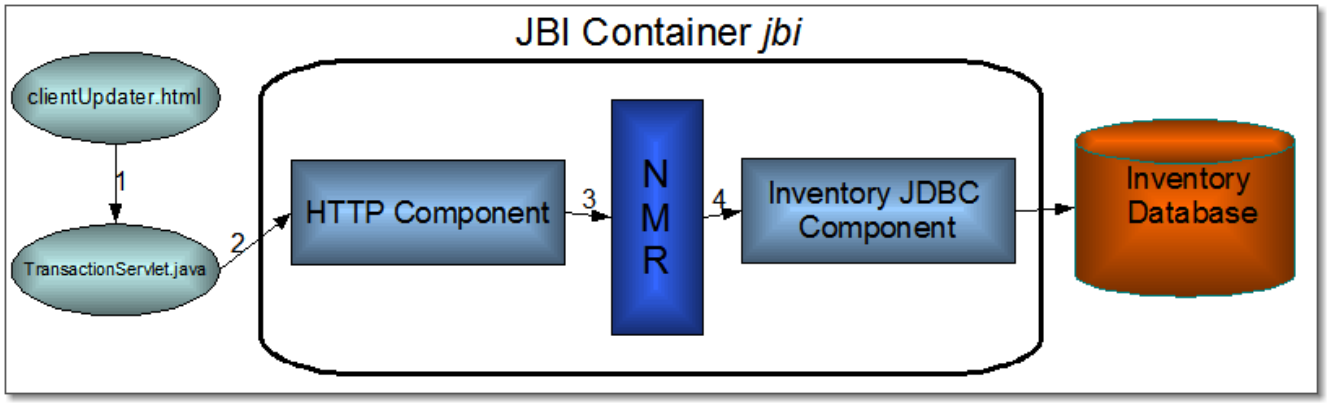


Figure 2: 24/7 Client System

Message Flow:

1. `Client.html` submits a form with the updated contents to the `TransactionServlet`.
2. `TransactionServlet` generates the XML that will be given to the JBI container. It either sends a single request, or loops and generates several requests.
3. `HTTP binding component` receives the HTTP request from the HTTP client and transforms it into a normalized message that is published to the Inventory JDBC component.
4. The Inventory `JDBC Component` receives the normalized message and executes the SQL request using the configured datasource.

Component Configuration:

1. The `TransactionServlet` creates a HTTP client that will send a request to the URL: http://localhost:8077/grocery-demo/JdbcService/
2. The specified URL is mapped to a HTTP binding component. This configuration is located in the `xbean.xml` of the `grocery-demo-http` module of the grocery demo and creates a `HTTP BC` mapped to the specified location URI.

```
<http:endpoint
    service="grocery:JdbcToResponseService"
    endpoint="grocery:HttpJdbcEndpoint"
    role="consumer"
    locationURI="http://localhost:8077/grocery-demo/JdbcService/"
    defaultMep="http://www.w3.org/2004/08/wsdl/in-out"
/>
```

- *service* – the target JBI service of this HTTP binding component. Messages received by this endpoint will be routed to the `JdbcToResponseService` endpoint.
- *endpoint* – identifies this `HTTP BC` in the JBI namespace.
- *role* – as a consumer the endpoint is able to accept requests from a component outside the JBI environment.
- *locationURI* – the URI this `HTTP BC` is mapped to.

- *defaultMep* – the message exchange pattern this endpoint will use. In this case, it will use the in-out MEP, which allows it to accept a request and provide a response.

3. The target of the `HTTP BC` is a custom JBI service called the `JdbcComponent` that is deployed in the lightweight container. The class inherits from a ServiceMix component that allows it to process message exchanges and either provides a response or not depending on the MEP. The `JdbcComponent` class provides access to the underlying datasource. The deployment configuration of the `JdbcComponent` is located in the `servicemix.xml` of the `grocery-demo-jdbc` module.

```
<sm:activationSpec
    componentName="JdbcToResponseComponent"
    endpoint="grocery:JdbcToResponseComponent"
    service="grocery:JdbcToResponseService">
    <sm:component>
    <bean class="org.logicblaze.soa.grocery.components.JdbcComponent">
       <property name="dataSource" ref="databaseServer"/>
       <property name="responseRequired" value="true"/>
    </bean>
    </sm:component>
</sm:activationSpec>
```

- *componentName/endpoint/service* – identifies this JBI service in the JBI namespace.
- *bean:class* – specifies the class name of this custom JBI component
- *bean:dataSource* – the spring configured datasource this component will use.
- *bean:responseRequired* – if true, the component will send a request/response if an empty record set is queried or updates were performed, else it will not send anything.

27/7 Monitoring System

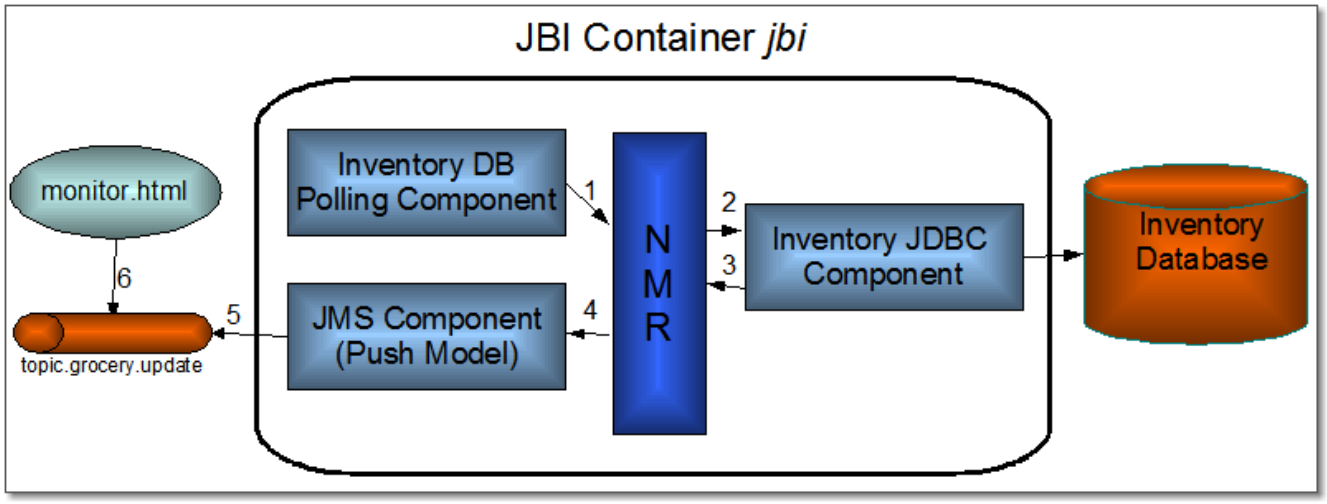


Figure 3: 27/7 Monitoring System

Message Flow:

1. A `polling component` is deployed which sends a JBI request for item updates every 2.5 seconds to the `Inventory JDBC component`.
2. The `JDBC Component` receives the update request, and queries all updated items.
3. The result set is then routed to the `JMS component`.
4. The `JMS component` receives the result set as normalized messages and converts it to a JMS text message.
5. The `JMS component` publishes the JMS text message to the JMS topic `topic.grocery.update`.
6. The `monitor.html` via the `AjaxServlet` receives the JMS message and updates the monitoring table with the selected items that have been updated.

Component Configuration:

1. The `Inventory DB Poller Component` is a custom `JBI component` that extends from a `polling component` provided by ServiceMix. It is deployed in the lightweight container, where it will constantly send a message to the Inventory `JDBC component` requesting the recent updates to the database. Its deployment configuration is located in the `servicemix.xml` of the `grocery-demo-poll` module.

```
<sm:activationSpec
    componentName="InventoryDbPoller"
    service="grocery:InventoryDbPollerService"
    interfaceName="grocery:InventoryDbPollerInterface"
    destinationService="grocery:JdbcToJmsService">
    <sm:component>
    <bean class="org.logicblaze.soa.grocery.components.InventoryDbPoller">
```

```
        <property name="period" value="2500"/>
    </bean>
    </sm:component>
</sm:activationSpec>
```

- *componentName/interfaceName/service* – identifies this JBI service in the JBI namespace.
- *destinationService* – the target JBI service of this component. This component will constantly send a message exchange to the `JdbcToJmsService`.
- *bean:class* – specifies the class name of this custom `JBI component`
- *bean:period* – the interval between each poll in milliseconds.

2. The target of the polling component is the same `JdbcComponent` that provides access to the Inventory Database, except that this component is configured to forward its result set to a JMS binding component. The deployment configuration of the `JdbcComponent` is located in the `servicemix.xml` of the `grocery-demo-jdbc` module.

```
<sm:activationSpec
    componentName="JdbcToJmsComponent"
    service="grocery:JdbcToJmsService"
    destinationService="jms-bind:MySqlDbUpdateJmsService">
    <sm:component>
    <bean class="org.logicblaze.soa.grocery.components.JdbcComponent">
        <property name="dataSource" ref="databaseServer"/>
    </bean>
    </sm:component>
</sm:activationSpec>
```

- *componentName/service* – identifies this JBI service in the JBI namespace.
- *destinationService* – the target JBI service of this component. This component will send the result set of its query to a configured `JMS component` identified by:

  `jms-bind:MySqlDbUpdateJmsService`
- *bean:class* – specifies the class name of this custom JBI component
- *bean:dataSource* – the spring configured datasource this component will use.

3. The result set of the query will be routed to a JMS component where it will publish the result as a JMS Message to a JMS Topic. The deployment configuration of the `JMS Component` is located in the `xbean.xml` of the `grocery-demo-jms` module.

```
<jms:endpoint
    service="jms-bind:MySqlDbUpdateJmsService"
    endpoint="mySqlDbUpdateJms"
    role="provider"
    destinationStyle="topic"
    jmsProviderDestinationName="topic.grocery.update"
    connectionFactory="#jmsFactory"
/>
```

- *service/endpoint* – identifies this `JMS BC` in the JBI namespace.

- *role* – as a provider the endpoint is able to provide JMS services from inside the JBI container to the outside environment. In this case, it allows JBI services to publish JMS messages to an external broker.
- *destinationStyle* – identifies the type of JMS destination whether topic or queue.
- *jmsProviderDestinationName* – the name of the JMS destination that the message will be published to.
- *connectionFactory* – the spring configured JMS connection factory that the binding component will use to connect to the external JMS broker.

Good Life Supplier Management System

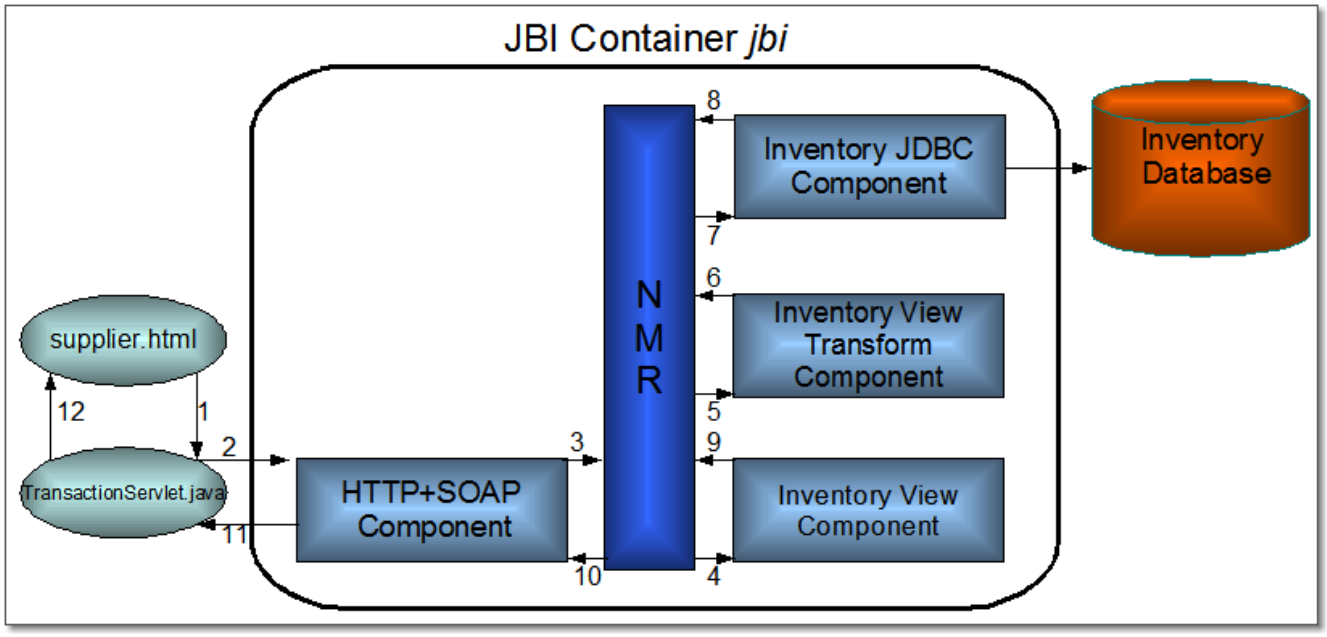


Figure 4: Good Life Supplier Management System

Message Flow:

1. `supplier.html` submits a form for checking supply quantities to the `TransactionServlet`.
2. `TransactionServlet` composes the SOAP message and sends it to the `HTTP + SOAP binding component`.
3. `HTTP + SOAP BC` receives the SOAP message. It then extracts the actual message content, transforms it into a normalized message, and sends it to the `Inventory View Component`.
4. The `Inventory View Component` receives the normalized message and routes it to the Inventory View Transform Component.
5. The `Inventory View Transform Component` receives the message where it is authenticated and transformed into a query statement message.
6. The transformed message is then sent back to the `Inventory View Component` as a response.
7. The `Inventory View Component` then routes the transformed message to the `Inventory JDBC component`, where the query is executed in the specified datasource.
8. The result of the query is then normalized and sent back to the `Inventory View Component` as a response.
9. The `Inventory View Component` then routes the final response back to the `HTTP + SOAP binding component` as its response.
10. The `HTTP + SOAP BC` receives the normalized message response, and transform it into a SOAP message and sent back as the response of the HTTP request.

11. The `TransactionServlet` receives the response and routes it to the `supplier.html`.
12. The `supplier.html` parses the response and displays the result in the Web page.

Component Configuration:

1. The `TransactionServlet` will create a HTTP client that will send a request to the URL: http://localhost:8077/grocery-demo/InventoryViewService/
2. The specified URL is mapped to a `HTTP + SOAP binding component`. This configuration is located in the `xbean.xml` of the `grocery-demo-http` module of the grocery demo and basically creates a `HTTP + SOAP BC` mapped to the specified location URI.

```
<http:endpoint
    service="grocery:InventoryViewService"
    endpoint="grocery:HttpInventoryViewEndpoint"
    role="consumer"
    soap="true"
    locationURI="http://localhost:8077/grocery-demo/InventoryViewService/"
    defaultMep="http://www.w3.org/2004/08/wsdl/in-out"
/>
```

- *service* – the target JBI service of this `HTTP + SOAP binding component`. Messages received by this endpoint will be routed to the `InventoryViewService` endpoint.
- *endpoint* – identifies this `HTTP + SOAP BC` in the JBI namespace.
- *role* – as a consumer the endpoint is able to accept requests from component outside the JBI environment.
- *soap* – indicates whether this `HTTP component` is SOAP enabled or not. In this case the component is SOAP enabled.
- *locationURI* – the URI this `HTTP + SOAP BC` is mapped to.
- *defaultMep* – the message exchange pattern this endpoint will use. In this case, it will use the in-out MEP, which allows it to accept a request and provide a response.

3. The `Inventory View Component` is a ServiceMix component that allows it to chain the response of a JBI component as the in message of the next component in the chain. The deployment configuration of the `Inventory View Component` is located in the `servicemix.xml` of the `grocery-demo-jdbc` module.

```
<sm:activationSpec
    componentName="InventoryViewComponent"
    endpoint="grocery:InventoryViewComponent"
    service="grocery:InventoryViewService">
    <sm:component>
    <util:ChainedComponent>
        <services>
            <qname>grocery:InventoryViewTransformService</qname>
            <qname>grocery:JdbcToResponseService</qname>
       </services>
  </util:ChainedComponent>
```

```
    </sm:component>
</sm:activationSpec>
```

- *componentName/endpoint/service* – identifies this JBI service in the JBI namespace.
- *util:ChainedComponent* – specifies the ServiceMix chain component as the JBI component that will be used.
- *services* – an ordered list of qualified names of JBI services that will be chained together.

4. The `Inventory View Transform Component` is a custom component that accepts a specific type of message format and transforms it to the query message format that is recognized by the `Inventory JDBC Component`. This allows the `Inventory View Transform Component` to perform simple authentication and limits the type of SQL statements that can be executed. The deployment configuration of the `Inventory View Transform Component` is located in the `servicemix.xml` of the `grocery-demo-jdbc` module.

```
<sm:activationSpec
    componentName="InventoryViewTransformComponent"
    endpoint="grocery:InventoryViewTransformComponent"
    service="grocery:InventoryViewTransformService">
    <sm:component>
    <bean class="org.logicblaze.soa.grocery.components.InventoryViewTransformComponent">
          <property name="auth">
               ...
          </property>
          <property name="role">
               ...
          </property>
    </bean>
    </sm:component>
</sm:activationSpec>
```

- *componentName/endpoint/service* – identifies this JBI service in the JBI namespace.
- *bean:class* – specifies the class name of this custom JBI component.
- *auth/role* – key-value pair of the allowed usernames and password that this component will recognize including the roles of each user.

5. The `Inventory JDBC Component` configuration used is the same as that described in the Component Configuration of the *24/7 Client System Section*.

Speed Trans Delivery Scheduling System

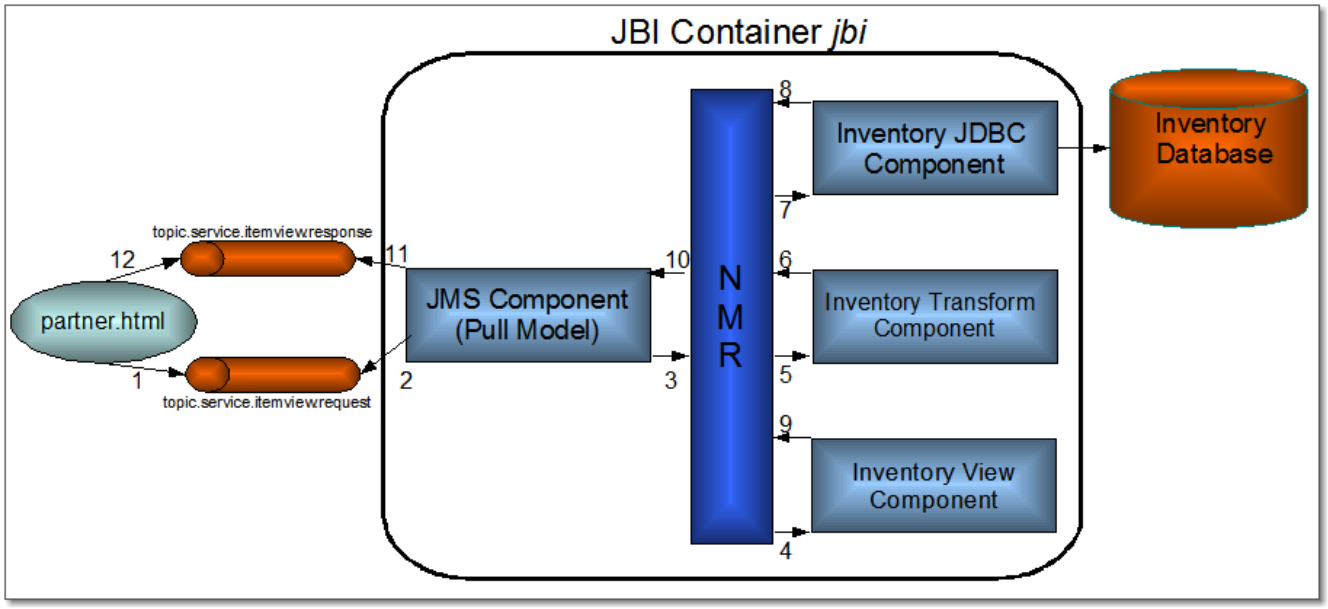


Figure 5: Speed Trans Delivery Scheduling System

Message Flow:

1. The `partner.html` via the `AjaxServlet` publishes an item availability request message to the JMS topic: `topic.service.itemview.request`.
2. The `JMS binding component` listens on the topic and receives the item request message and transforms it into a normalized message.
3. The `JMS BC` sends the normalized message to the `Inventory View Component` for processing.
4. The `Inventory View Component` receives the normalized message and routes it to the `Inventory View Transform Component`.
5. The `Inventory View Transform Component` receives the message where it is authenticated and transformed into a query statement message.
6. The transformed message is then sent back to the `Inventory View Component` as a response.
7. The `Inventory View Component` then routes the transformed message to the Inventory `JDBC component`, where the query is executed in the specified datasource.
8. The result of the query is then normalized and sent back to the `Inventory View Component` as a response.
9. The `Inventory View Component` then routes the final response back to the `JMS binding component` as its response.
10. The `JMS BC` receives the normalized message response and transforms it into a JMS message.

11. The `JMS BC` sends the message to the destination specified in the `JMSReplyTo` header of the message. In this case, the destination is the JMS topic: `topic.service.itemview.response.`
12. The `partner.html` via the `AjaxServlet` receives the message and updates the item availability table in the web page.

Component Configuration:

1. The `JMS Component` allows the partner application to interact with the Inventory DB via JMS by publishing and subscribing to specific JMS topics implementing a request-response model. The deployment configuration of the `JMS Component` is located in the `xbean.xml` of the `grocery-demo-jms` module.

```
<jms:endpoint
    service="grocery:InventoryViewService"
    endpoint="InventoryViewComponent"
    defaultMep="http://www.w3.org/2004/08/wsdl/in-out"
    role="consumer"
    destinationStyle="topic"
    jmsProviderDestinationName="topic.service.itemview.request"
    connectionFactory="#jmsFactory"
/>
```

   - *service/endpoint* - the target JBI service of this `JMS binding component`. JMS messages received by this endpoint will be routed to the `InventoryViewService` endpoint.
   - *defaultMep* – the message exchange pattern this endpoint will use. In this case, it will use the in-out MEP, which allows it to accept a request and provide a response.
   - *role* – as a consumer, this `JMS BC` is able to listen to an external JMS topic and process requests directed to it. In this case, the `JMS BC` will listen for item availability requests from the JMS topic: `topic.service.itemview.request.`
   - *destinationStyle* – identifies the type of JMS destination whether topic or queue.
   - *jmsProviderDestinationName* – the name of the JMS destination that the `JMS BC` will listen from.
   - *connectionFactory* – the spring configured JMS connection factory that the binding component will use to connect to the external JMS broker.
2. The `Inventory View Component` configuration used is the same as that described in the Component Configuration of the *Good Life Supplier Management System* Section.
3. The `Inventory View Transform Component` configuration used is the same as that described in the Component Configuration of the *Good Life Supplier Management System* Section.
4. The `Inventory JDBC Component` configuration used is the same as that described in the Component Configuration of the *24/7 Client System* Section.

3. Non-Ajax Implementation

This section shows the details of the (non-preferred) approach without Ajax.

3.1. Running the Demonstration

To run the Non-Ajax demo perform the following steps:

1. Go to `http://localhost:8080/grocery-demo/`
2. Click the "Non-Ajax Demo" link. This will show you the inventory of the "24/7 Grocery" store.
3. Click the link to open the "Transaction Generator". You may chose to specify a values and submit a single transaction or send 10 random transactions. Please watch your "Inventory Monitoring System" window to see the results of this action.
4. Click the link to open the "Supplier Page". Enter a value in the "Stock Limit" field. Click "Search" to see what items have gone below the limit.
5. Click on the link to open the "Partner Page". Select as many check boxes as you wish in "Grocery Items" column and click "Check Items Availability" to view the availability of each individual grocery item.

3.2. Details

The functionality of the Ajax and the non-Ajax Demonstrations are essentially the same, but the execution has two very distinct feels. The Ajax demonstration (the one we prefer) is cleaner in its feel when compared to the non-Ajax demonstration. Notice how the non-Ajax demonstration has to refresh the whole web page. The Ajax demonstration just refreshes the numbers.

4. Additional Resources

For more information about MySQL please see http://www.mysql.com/.

MySQL Documentation for 5.0 - http://dev.mysql.com/doc/refman/5.0/en/index.html

For more information about Derby please see http://db.apache.org/derby/

Derby Documentation - http://db.apache.org/derby/manuals/index.html

Open Source Database Comparison - http://www.devx.com/dbzone/Article/29480?trk=DXRSS_DB

For more information on lightweight components please see http://www.servicemix.org/What+is+a+lightweight+component.

For more information on deploying lightweight components please see http://docs.codehaus.org/display/SM/Deploying+Lightweight+Components+Tutorial.