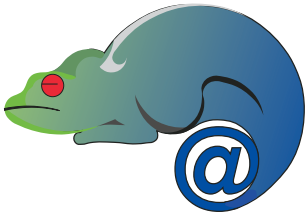
ADAPT
IST-2001-37126

Middleware Technologies for Adaptive and Composable Distributed Components

Demonstrator Basic Service



Deliverable Identifier: D18
Delivery Date: 23rd March 2005
Classification: Public Circulation
Authors: Stuart Wheater
Document version: 1.0 23rd March 2005

Contract Start Date: 1st September 2002
Duration: 36 months
Project coordinator: Universidad Politécnica de Madrid (Spain)
Partners: Universitá di Bologna (Italy), ETH Zürich (Switzerland), McGill University (Canada), Universitá degli Studi di Trieste (Italy), University of Newcastle (UK), Arjuna Technologies Ltd (UK)

Project funded by the European Commission under the Information Society Technologies Programme of the 5th Framework (1998-2002)

Table of Contents

1 Introduction

The purpose of this Demonstrator Basic Service deliverable is to provide a software system that will allow the easy demonstration of the capabilities of the BS Middleware deliverable. The BS Middleware uses replication as the key technique for providing adaptability; it is the adaptability to failure provided by the BS Middleware that the demonstrator will illustrate. This deliverable also forms a milestone on the progress towards final Adapt Demonstrator deliverable, which will demonstrate Basic Service and Composite Service support, and their integration.

This deliverable is based around part of the WS-I Sample Application Specification [1][2][3]. This specification details a design for a Supply Management System, based on Web services. We have implemented some of the services that make up the design using J2EE technologies, such as Session Beans and Entity Beans, on top of the BS Middleware. The BS Middleware supports the replication of these technologies. A set of web pages has also been constructed, to drive and monitor the demonstrator, and these act as a client for the replicated Supply Management service.

The part of the WS-I Sample Application Specification that makes up this demonstrator is one of the interactions between the Warehouse of a Retailer and a Manufacturer. This interaction consists of the Warehouse service sending a purchase order to a Manufacturer service, which is checked and if correct acknowledged. This is followed by, at some later point, the Manufacturer sending a shipment notice to Warehouse. This interaction is illustrated in figure 2; related interactions that don't achieve the desired outcome are illustrated in figures 1 and 3.

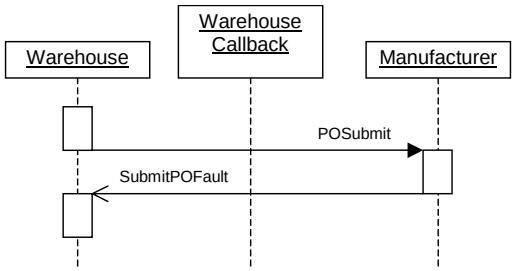


Figure 1: Incorrect purchase order interaction.

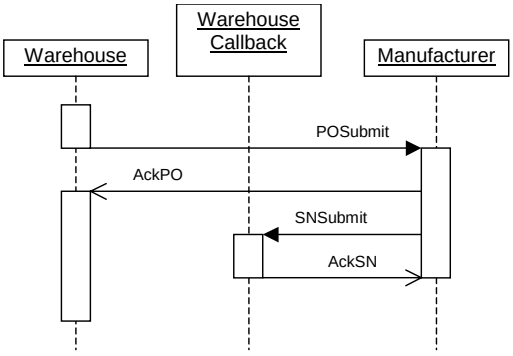


Figure 2: Correct purchase order and shipment notice interaction.

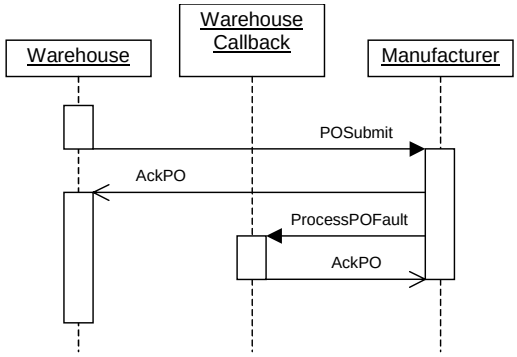


Figure 3: Correct purchase order, but incorrect shipment notice interaction.

In the demonstrator the Manufacturer is implemented as dummy composite service, being a Java application rather than a workflow driven process, which performs it desired purpose by making a series of invocation on basic service. This is described in greater detail in section 4.

2 Setting up the Demonstrator

The setting up of the demonstrator is a complicated process, so it is advised to try to stick as closely as possible to the instruction given below. The demonstrator can be deployed and configured to require between one and five machines, or more if more than two basic service replicas are desired. The follow instructions assume that four machines will be used; this can actually be reduced to three if the database is collocated on one of the other machines. An illustration of the relationship between the processes (operating system processes, not workflow processes) and machines that will be described in these setup instructions is given is figure 4.

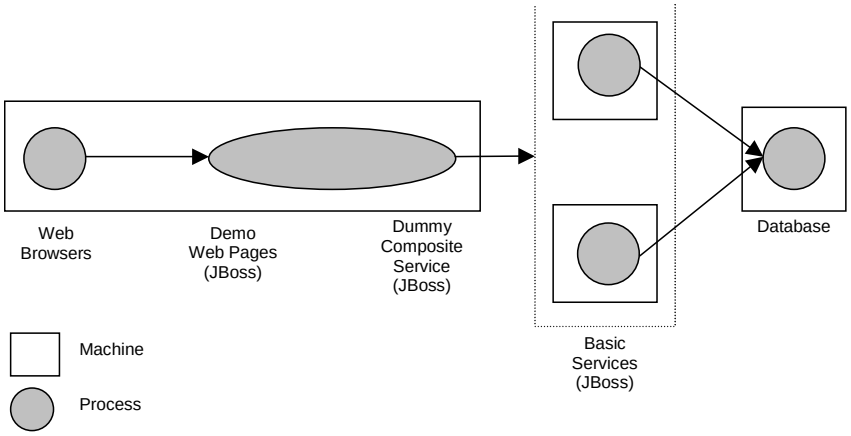


Figure 4: relationship between the processes and machines.

2.1 Required software

The setting up this demonstrator requires the following software:

- A web browser
- Demonstrator Basic Service-1.0
  - available via http://adapt.ls.fi.upm.es/Downloads.htm
- Sun's JDK-1.4.2
  - available via http://java.sun.com/j2se/1.4.2/download.html
- Apache's ant-1.6.2
  - available via http://ant.apache.org/bindownload.cgi
- Apache's BCEL-5.1
  - available via http://jakarta.apache.org/bcel/
- JBoss-3.2.3
  - available via http://sourceforge.net/projects/jboss/
- jboss-adapt-framework-1.0-1
  - available via http://sourceforge.net/projects/j2ee-adapt/
- jboss-adapt-replication-sib-1.0
  - available via http://sourceforge.net/projects/j2ee-adapt/
- Spread-3.17.3
  - available via http://www.spread.org/

- Database
  - PostgreSQL 8.0
    - available via http://www.postgresql.org/download/
  - Oracle 10g,
    - available via http://www.oracle.com/
  - MySQL 4.1
    - available via http://dev.mysql.com/downloads/
  - IBM's DB2 V8.2
    - available via http://www.ibm.com/software/data/db2/

The operating systems of the machines on which the demonstrator will be run should not be signification, as long as the required software is supported. The demonstrator has primarily been developed and tested on Linux and Windows2000, but this does not preclude the use of other operating systems.

2.2 Setup steps

As was indicated above these setup instructions are intended of use with four machines. Throughout the following steps these machine will be refereed to as "demo+dcs", "bs1", "bs2" and "db", respectively referring to demo and dummy composite service, basic services 1, basic service2, and database.

1. Install Sun's JDK on the machines: "demo+dcs", "bs1" and "bs2".

2. Install Apache's ant on the machines: "demo+dcs", "bs1" and "bs2".

3. Copy BCEL's bcel.jar to the lib directory of the ant installations on the machines: "demo+dcs", "bs1" and "bs2".

4. Install JBoss on the machines: "demo+dcs", "bs1" and "bs2".

5. Install and configure Spread on machines "bs1" and "bs2". To configure Spread consult the Readme.txt within the distribution. Note that the spread.conf should be the same on "bs1" and "bs2".

6. Ensure the environment variable JBOSS_HOME is set to the root of the JBoss installation.

7. Install jboss-adapt-framework on the machines: "demo+dcs", "bs1" and "bs2", by extracting the jboss-adapt-framework source, and change into that directory. Within this directory issue the command 'ant'.

8. Install jboss-adapt-replication-sib on the machines: "demo+dcs", "bs1" and "bs2", by extracting the jboss-adapt-replication-sib source, and change into that directory. Within this directory issue the command 'ant'.

9. Configure jboss-adapt-replication-sib on the "demo+dcs" machine. This is done by setting the code attribute of the clientcomponentmonitor tag to org.eu.adapt.bs.rep.sib.alg.txcommitting.WSClientReplicationManager in the file <jboss_home>/server/adapt/config/clientcomponentmonitor-config.xml.

10. Configure jboss-adapt-replication-sib on the machine: “bs1” and “bs2”. This is done by copying the componentmonitor-config.xml file from the distribution <jboss_home>/server/adapt/config/componentmonitor-config.xml and making the changes described within the file.

11. Install the desired database system on the “db” machine (which could be “demo+dcs”, if it is a powerful machine).

12. Configure “adapt” JBoss’s profiles, on the machines “bs1” and “bs2”, to use the installed database. The easiest way this can be done is the remove the <jboss_home>/server/adapt/deploy/hsqldb-ds.xml file, and copying the appropriate “-ds.xml” file for the chosen database to <jboss_home>/server/adapt/deploy, then changing the jndi-name to DefaultDS. Note: other changes could be required by JBoss, for example adding jar files.

13. Ensure the environment variable JBOSS_PROFILE to “adapt”.

14. Build “Demonstrator Basic Service” on the machines: “demo+dcs”, “bs1” and “bs2”. Extract the “Demonstrator Basic Service” source, on each machine, and change into that directory. Set the properties “basicservices.host”, “basicservices.port”, “compositeservices.host”, “compositeservices.port”, “demoservices.host”, “demoservices.port” in the “build.xml file. Within this directory issue the command ‘ant compile’.

15. Install “Demonstrator Basic Service” on the machines: “bs1” and “bs2”. Change into the “Demonstrator Basic Service” source directory. Within this directory issue the command ‘ant bs_deploy’.

16. Install “Demonstrator Basic Service” on the “demo+dcs” machine. Change into the “Demonstrator Basic Service” source directory. Within this directory issue the command ‘ant cs_deploy demo_deploy’.

3 Running the Demonstrator

To run the Demonstrator the following processes/services need to be running:

1. The chosen database on the machine “db”.

2. Spread on the machines “bs1” and “bs2”.

3. JBoss on the machines “demo+dcs”, “bs1” and “bs2”.

If the demonstrator has been correctly installed, pointing a web browser at the page http://<demo+dcs>:8080/index.html will result in the page in figure 5 being displayed.

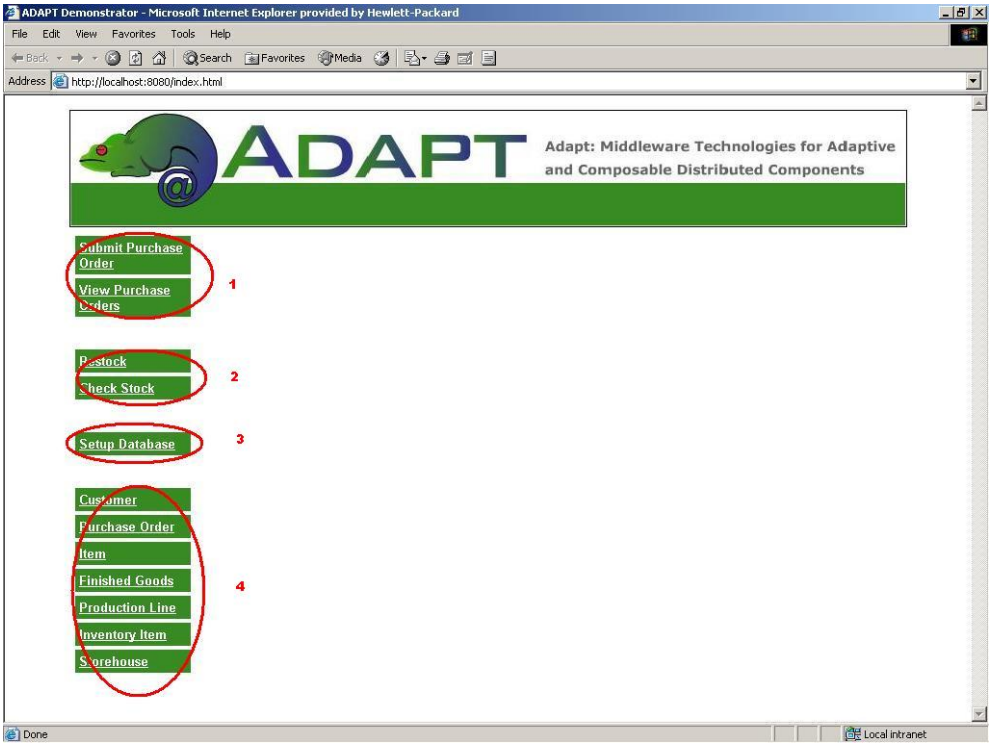


Figure 5: initial web page plus markup.

In figure 5, the four major groups of "buttons" which are used to interact with the demonstrator are indicated by red ovals, the purposes of these groups of "buttons" are:

Groups 1: These buttons can be used to submit and view the status of purchase orders. The orders placed by the "Submit Purchase Order" button is fixed at 10 units of "White Paper" for "The Financial Time". The status of past and outstanding purchase orders can be viewed by using the "View Purchase Orders" button.

Groups 2: These buttons can be used to cause a restocking event and a checking if the purchase orders can be fulfilled. The "Restock" button causes a fixed restock of 15 units of "White Paper" to be produced from production line "Production Line-1".

Groups 3: This button, "Setup Database", will clear the contents of the database and recreate the content than are the basis of the demonstration.

Groups 4: This group of buttons are used to inspect the contents of the database. Via these buttons the database entities, which are: customers, purchase orders, items, finished goods, production line, inventory items and storehouses, can be listed, there values inspected, and there relationships navigated. These buttons are not used in the demonstrator scenario, as there primary purpose is for debugging.

3.1 Scenario

The purpose of this scenario is to show fail-over of a basic service invocation from primary service instance to a secondary service instance.

The following assumes that the processes/services listed at the beginning of section 3 are running.

Step 1) Press the "Reset Database" button on the web page. This should result in a web page containing the message "**Manufacturer Setup:** Successful".

Pressing this button has caused a servlet in the "Demo Web Pages" process to make a web service invocation to the basic service primary. This invocation is implemented by a stateful session bean, which makes a series of invocation on entity beans (and there home interfaces) to firstly remove all states associated with the entity beans of the demonstrator that are present in the database, then secondly create the states and relationships in the database that are needed to perform the demonstration.

Step 2) Press the "Submit Purchase Order" button **two times** on the web page. This should result finally in a web page containing the message "Purchase Order Number: ON2, Status: Acknowledged". "ON2" being the purchase order number of the final purchase order submitted.

Each press of this button has caused a servlet in the "Demo Web Pages" process to make a web service invocation, of sumbitPO, to the dummy manufacturer composite service. This invocation is implemented by a JAX-RPC style web service, which makes two further web-service invocations this time on the basic service primary instance. These invocations are checkPurchseOrder and addPurchaseOrder, the interactions between the dummy manufacturer composite services, the basic services and their entity beans are illustrated in figure 6.

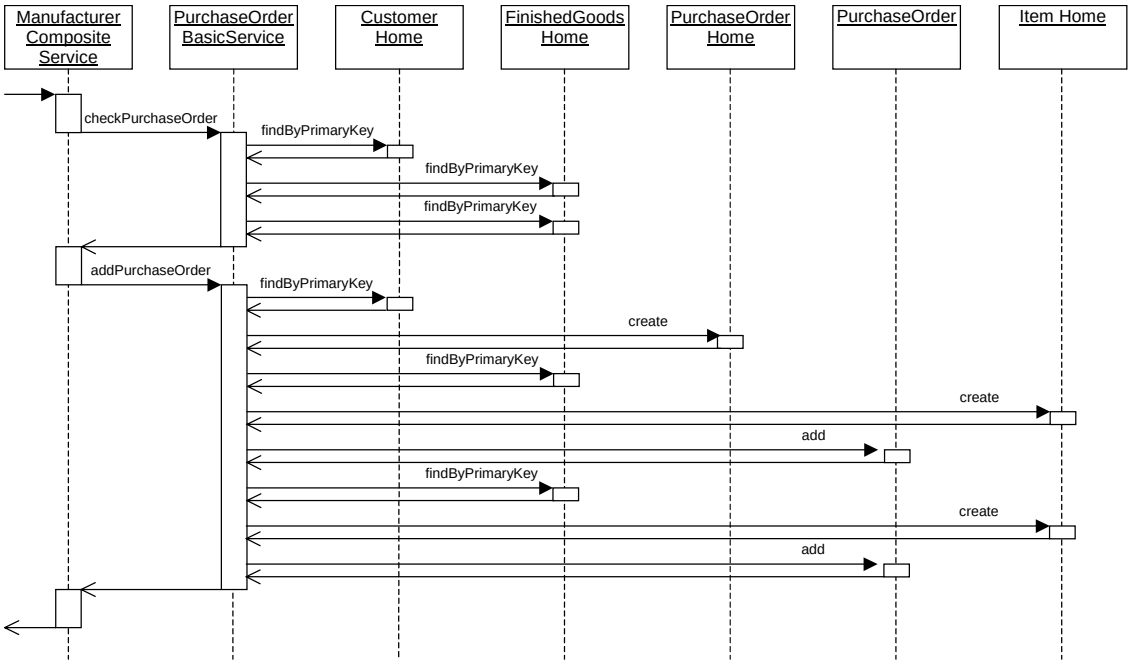


Figure 6: interaction diagram for submit purchase order invocation on the manufacturer.

Step 3) Press the "View Purchase Orders" button on the web page. This should result in a web page containing the following:

Figure 7: expected results of step 3.

Pressing this button has caused a servlet in the "Demo Web Pages" process to generate, from information contained within the "Demo Web Pages" process, a list of all know purchase orders and there status. The status will depend on if the submit purchase order invocation's response contained an acknowledgment ("Acknowledged" or "Unacknowledged"), and if a call back has been performed to indicated shipment ("Complete") or purchase order fault ("Rejected").

Step 4) Press the "Restock" button on the web page. This should result in a web page containing the following:

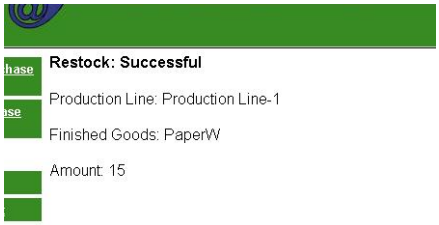


Figure 8: expected results of step 4.

Pressing this button has caused a servlet in the "Demo Web Pages" process to make a web service invocation to the basic service primary, to simulate a restock event. This invocation is implemented by a stateful session bean, which makes a series of invocation on entity beans (and there home interfaces) to increase the quantity, by 15, in the inventory item entry, for "White Paper", associated with the storehouse for "Production Line-1".

Because each purchase order is for 10 units and restocks are of 15 units, the first restock can satisfy a single purchase order.

Step 5) Press the "Check Stock" button on the web page. This should result in a web page containing something like the following:

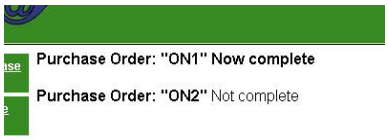


Figure 9: expected results of step 5.

Pressing this button has caused a servlet in the "Demo Web Pages" process to make a web service invocation to the basic service primary, to initiate a scan of the outstanding items in purchase orders, and see if they can be fulfilled. Any purchase orders which are fulfilled will cause a callback to the "Demo Web Pages" with the shipment notice. This original invocation is implemented by a stateful session bean, which makes a series of invocation on entity beans (and there home interfaces) this results in purchase order items being associated with a storehouse containing the appropriates finish goods.

Step 6) Press the "Restock" button on the web page. This should result in a web page containing the same as figure 8, of step 4.

Because each purchase order is for 10 units and restocks are of 15 units, the second restock will satisfy two further purchase orders.

Step 7) Terminate the JBoss instance on the machine "bs1".

This means that the primary basic service instance will now be unable to service requests.

Step 8) Press the "Submit Purchase Order" button **two times** on the web page. This should result finally in a web page containing the message "Purchase Order Number: ON4, Status: Acknowledged". "ON4" being the purchase order number of the final purchase order submitted.

Because the primary basic service instance has been terminated these requests cannot be done by that service, so will be automatically be switched to be handled by the secondary basic service instance.

Step 9) Press the "Check Stock" button on the web page. This should result in a web page containing something like the following:

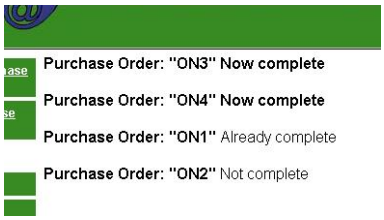


Figure 10: expected results of step 9.

This request will be sent automatically to the secondary basic service instance.

Step 10) Press the "View Purchase Orders" button on the web page. This should result in a web page containing the following:

Figure 11: expected results of step 10.

Pressing this button has caused a servlet in the "Demo Web Pages" process to generate, from information contained within the "Demo Web Pages" process, a list of all know purchase orders and there status.

4 Internals of the Demonstrator

As the setup details of the demonstrator implied, demonstrator the demonstrator's internal structure is fairly complicated. It is the intention of this section to explain the demonstrator's internal details in more detail, so a clearer understanding of what the demonstrator is illustrating can be achieved. The internal interactions of the demonstrator are illustrated in figure 12.

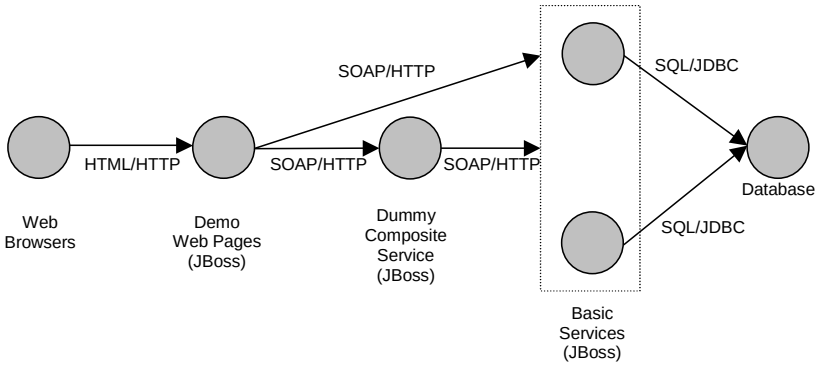


Figure 12: interactions within the demonstrator.

These interactions will how be described starting from right of figure 12 moving left:

The *Basic Services* interact with *Database* using a JDBC connection over which SQL commands are sent and result sets returned. The sources of the database interactions are the *Entity Beans* that manage the persistent state of the *Basic Services*.

The *Dummy Composite Service* and *Demo Web Pages* interact with the *Basic Services* via SOAP over HTTP. These interactions will be automatically switch from the primary to the secondary if the primary fails; this is done by the BS middleware. The details of the approach taken to achieve this are detailed in [4].

In the demonstrator the *Demo Web Pages* act as a client application to a composite service, in this demonstrator the composite service, is just a dummy, because it is implemented as a Java application rather than a workflow driven process. The interaction between the *Demo Web Pages* and the *Dummy Composite Service* is performed by as SOAP over HTTP, these invocation are not done via the BS middleware, because composite service make service invocations, which is precluded by the restrictions of basic services.

The *Web Browser* and *Demo Web Pages* interact with simple HTML over HTTP.

4.1.1 Basic Service Persistent Store Design

To ensure that the database interactions are as realistic as possible the persistent storage design follows the outline of the persistent data model given in the WS-I Sample Application [1][2][3]. This is reproduced in figure 13.

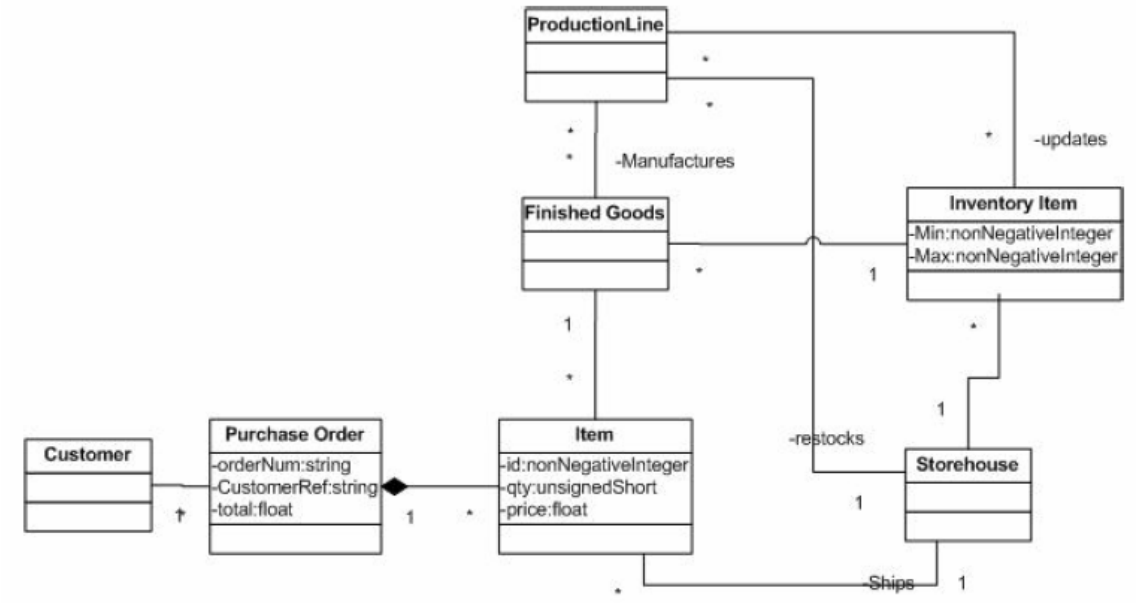


Figure 13: WS-I Sample Application persistent data model.

The persistent data model given in the WS-I Sample Application is not explained and contains some relationships that appear to be incorrectly specified. As a result the persistent data model used is slightly different, but it is hoped will also give a realistic interaction pattern with the database. This modified persistent data model is illustrated in figure 14.

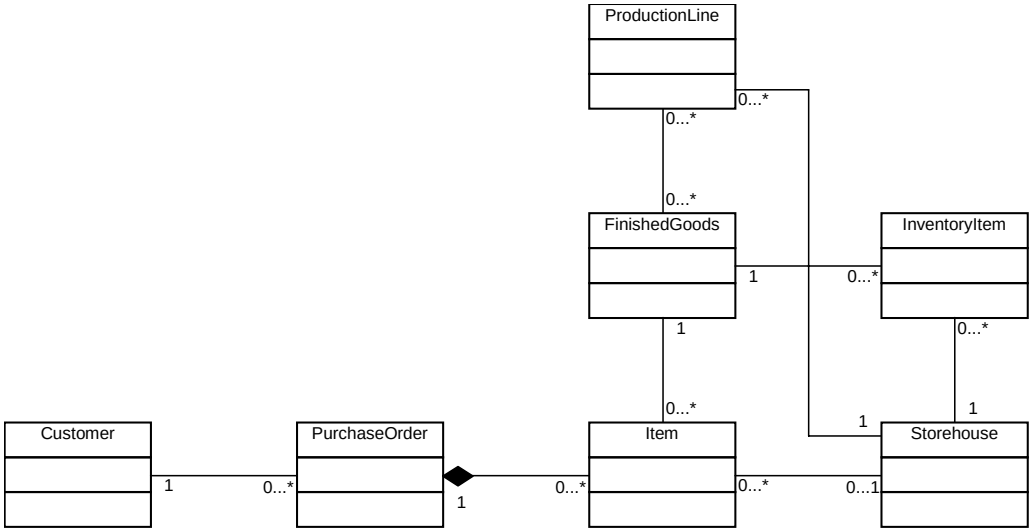


Figure 14: Demonstrator persistent data model.

5 References

[1] "Supply Chain Management Sample Application Architecture", WS-I, Version 1.0, December 2003 (http://www.ws-i.org/SampleApplications/SupplyChainManagement/2003-12/SCMArchitecture1.01.pdf).

[2] "Supply Chain Management Use Case Model", WS-I, Version 1.0, December 2003 (http://www.ws-i.org/SampleApplications/SupplyChainManagement/2003-12/SCMUseCases1.0.pdf).

[3] "WS-I Basic Profile Version 1.0a", WS-I (http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.html).

[4] "BS Middleware Platform", A. Bartoli, R. Jiménez-Peris, B. Kemme, S. Patarin, M. Patiño-Martínez, F. Perez, M. Prica, J. Salas, J. Vučković, H. Wu and S. Wu. (http://adapt.ls.fi.upm.es/deliverables/d13.pdf)