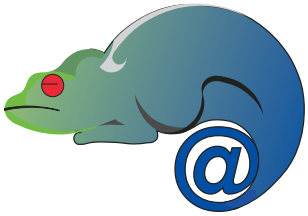
ADAPT
IST-2001-37126

Middleware Technologies for Adaptive and Composable Distributed Components

Demonstrator



Deliverable Identifier: D20
Delivery Date: 30th September 2005
Classification: Public Circulation
Authors: Stuart Wheater
Document version: 1.0 29th September 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 deliverable is to provide a software system that will allow the easy demonstration of the capabilities of the CS and BS Middleware deliverable. The ADAPT platform uses Basic Service replication as the key technique for providing adaptability to failure. The ADAPT platform uses the capabilities of the composite service execution engine to adapt to semantic and syntactic differences between services. The purpose of the ADAPT demonstrator is to show the inter-working of basic and composite service support.

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, some as Basic Service and others as Composite Services. The Basic Services 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. The Composite Services are implemented as workflow processes, which are executed by JOpera. 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.

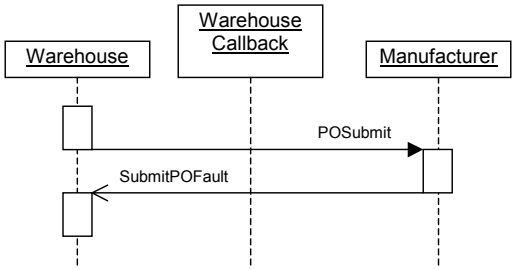


Figure 1: Incorrect purchase order interaction.

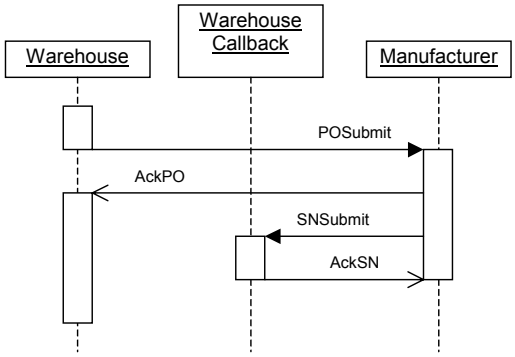


Figure 2: Correct purchase order and shipment notice interaction.

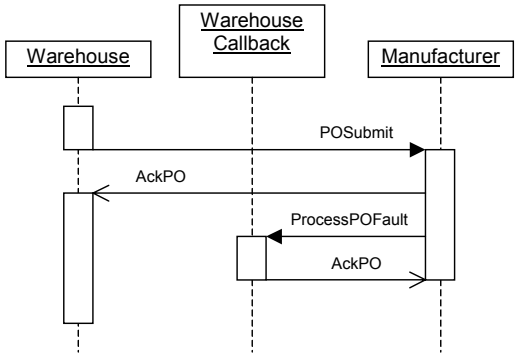


Figure 3: Correct purchase order, but incorrect shipment notice interaction.

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 needs to be deployed and configured on at least seven machines. The follow instructions assume that seven machines will be used; for simplicity the basic service are not replicated; this can actually be reduced to six 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.

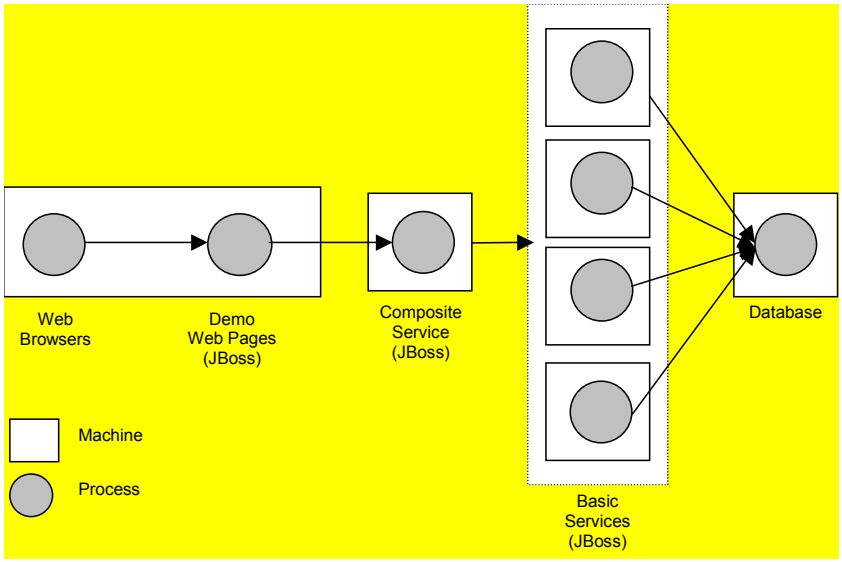


Figure 4: relationship between the processes and machines.

2.1 Required software

The setting up this demonstrator requires the following software:

- A web browser
- Demonstrator-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-1.6.5
  - 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-1.1-1
  - available via http://sourceforge.net/projects/j2ee-adapt/
- jboss-adapt-replication-sib-1.1
  - available via http://sourceforge.net/projects/j2ee-adapt/
- adapt-sensor-1.2
  - available via http://sourceforge.net/projects/j2ee-adapt/

- Spread-3.17.3
    - available via http://www.spread.org/
- PostgreSQL 8.0
    - available via http://www.postgresql.org/download/

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 seven machines. Throughout the following steps these machine will be refereed to as “demo”, “cs”, “r_bs”, “ma_bs”, “mb_bs”, “mc_bs” and “db”, respectively referring to demo, composite service, retailer basic services, manufacture A basic service, manufacture B basic service, manufacture C basic service and database.

1. Install Sun’s JDK on the machines: “demo”, “cs”, “r_bs”, “ma_bs”, “mb_bs”, “mc_bs” and “db”

2. Install Apache’s ant on the machines: “demo”, “cs”, “r_bs”, “ma_bs”, “mb_bs”, “mc_bs” and “db”.

3. Copy BCEL’s bcel.jar to the lib directory of the ant installations on the machines: “demo”, “cs”, “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”.

4. Install JBoss on the machines: “demo”, “cs”, “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”.

5. Install and configure Spread on machines “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”. To configure Spread consult the Readme.txt within the distribution.

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

7. Install jboss-adapt-framework on the machines: “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”, 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: “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”, by extracting the jboss-adapt-replication-sib source, and change into that directory. Within this directory issue the command ‘ant’.

9. Install adapt-sensors on the machines: “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”, by extracting the adapt-sensor source. Within this directory make the required changes to the build.xml file, as specified in the README file then, issue the command ‘ant’. Then change into that directory adapt-sensors-admin-1.2 then adapt-sensors-admin-1.2.

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

11. Install the desired database system on the “db” machine (which could be “demo” or “cs”, if they are powerful machines).

12. Configure “adapt” JBoss’s profiles, on the machines “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”, 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” on the machines: “demo”, “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”. Extract the “Demonstrator Basic Service” source, on each machine, and change into that directory. Set the properties “demo.basicservices.host”, “demo.basicservices.port”, … etc. in the “build.xml file. Within this directory issue the command ‘ant compile’.

15. Install “Demonstrator” on the machines: “demo”. Change into the “Demonstrator” source directory. Within this directory issue the command ‘ant demo_deploy’.

16. Install “Retailer Basic Service” on the “r_bs” machine. Change into the “Demonstrator” source directory. Within this directory issue the command ‘ant retailer_bs_deploy’.

17. Install “Manufactures Basic Service” on the “ma_bs”, “mb_bs” and “mc_bs” machine. Change into the “Demonstrator” source directory. Within this directory issue the command ‘ant manufacturer_a_bs_deploy’ ‘ant manufacturer_b_bs_deploy’ ‘ant manufacturer_c_bs_deploy’, respectively.

18. Install JOpera on the machine “cs”

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 “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”.

3. JBoss on the machines “demo”, “r_bs”, “ma_bs”, “mb_bs” and “mc_bs”.

4. JOpera on the machines “cs”.

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.

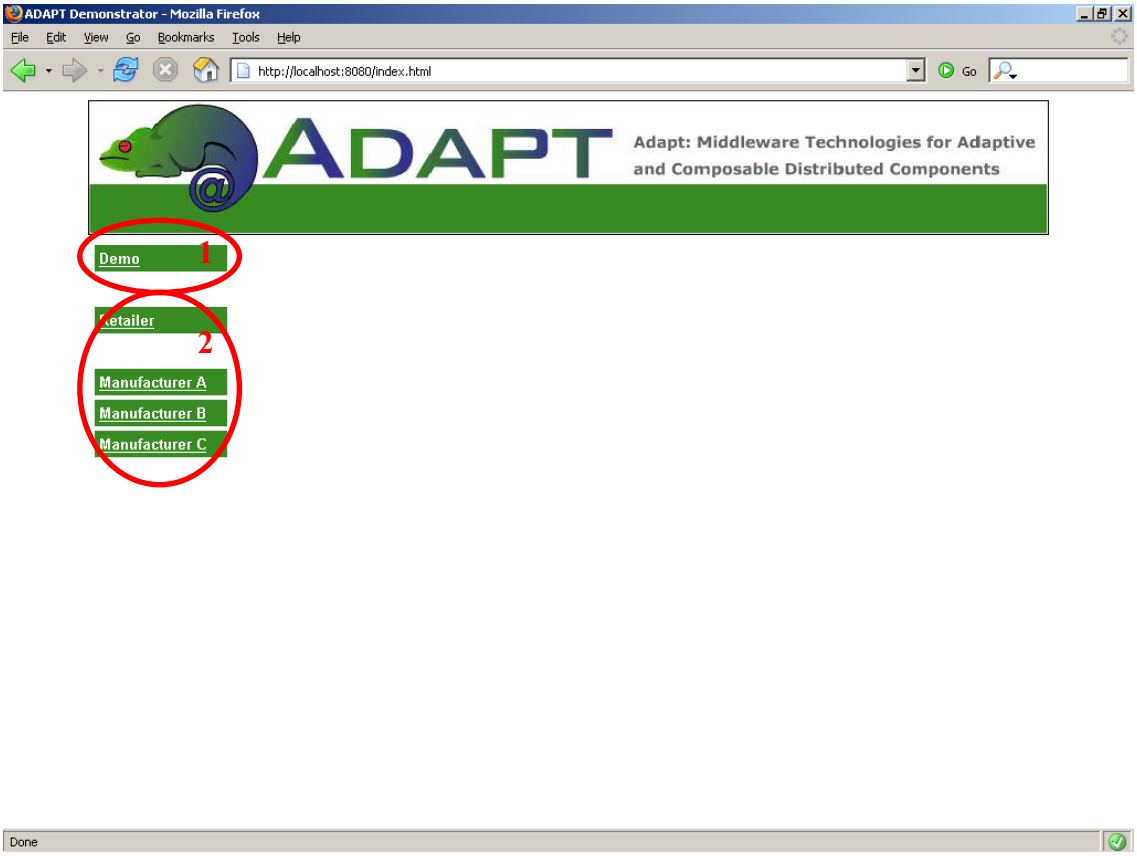


Figure 5: initial web page plus markup.

In figure 5, the two groups of “buttons” which are used to interact with the demonstrator are indicated by red ovals, the purposes of these groups of “buttons” are:

Group 1: Takes you to the demonstrator controls

Groups 2: 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.

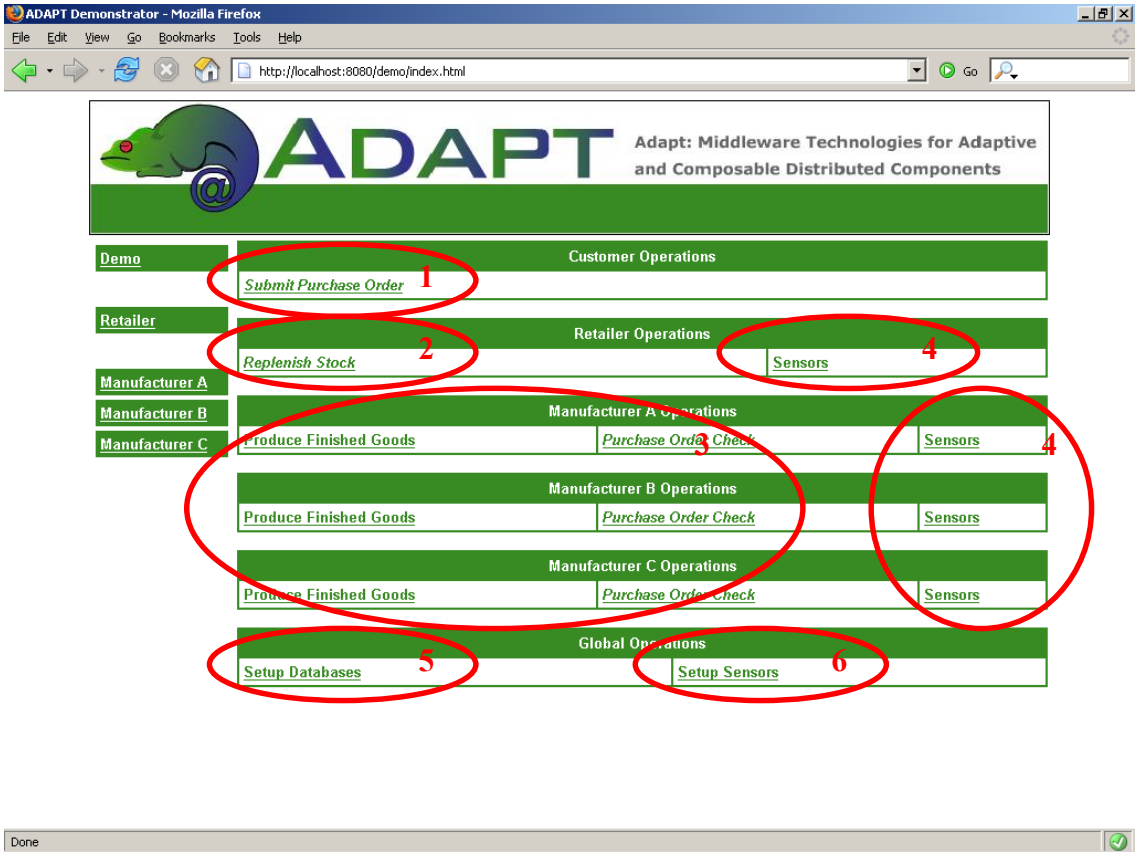


Figure 6: main demo web page plus markup.

Groups 1: These buttons can be used to submit and view the status of purchase orders.

Groups 2: These buttons can be used to cause a replenishment of the stock of the retailer.

Groups 3: These buttons can be used to cause a produce finished goods event and a checking if the purchase orders can be fulfilled.

Groups 4: This button, "Sensors", allow inspection of the sensor information associated with the services.

Groups 5: This button, "Setup Database", will clear the contents of the database and recreate the content than are the basis of the demonstration.

Groups 6: This button, "Setup Sensors", will clear sets up all sensors.

3.1 Scenario

The purpose of this scenario is to show the integration of a composite and basic service, within the same application.

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 a table showing the databases that have been successful setup.

Pressing this button has caused a servlet in the "Demo Web Pages" process to make series of web service invocations to the basic service primary. These invocation are 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 will show the web page below. Order 1000 units of "Black Ink, 500l Black Ink" by entering 1000 into the input field, then press "Submit".

Catalogs Event	
Black Ink, 500l Black Ink	0
Blue Ink, 100l Blue Ink	0
Red Ink, 100l Red Ink	0
Yellow Ink, 100l Yellow Ink	0
White Paper, 100m2 White Paper 25g/m2	0
Pink Paper, 100m2 Pink Paper 25g/m2	0
Submit	

Figure 7, submit purchase order form

This order will be "Rejected", this is because the retailer does not have any stock of "Black Ink".

Each press of this button has caused a servlet in the "Demo Web Pages" process to make a web service invocation to the retailer composite service. This invocation is implemented by a JOpera web service, which makes further web-service invocations this time on the basic service primary instance.

Step 3) Press the "Replenish Stock" button will show the web page similar to the w-eb-page above. Order 10000 units of "Black Ink, 500l Black Ink" by entering 10000 into the input field, then press "Submit".

This order will be "Acknowledged", this means that an order has been place with the manufacture, but stock may not have been received.

Pressing the button has caused a servlet in the "Demo Web Pages" process to make a web service invocation, of sumbitPO, to the manufacturer composite service. This invocation is implemented by a JOpera 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 manufacturer composite services, the basic services and their entity beans are illustrated in figure 8.

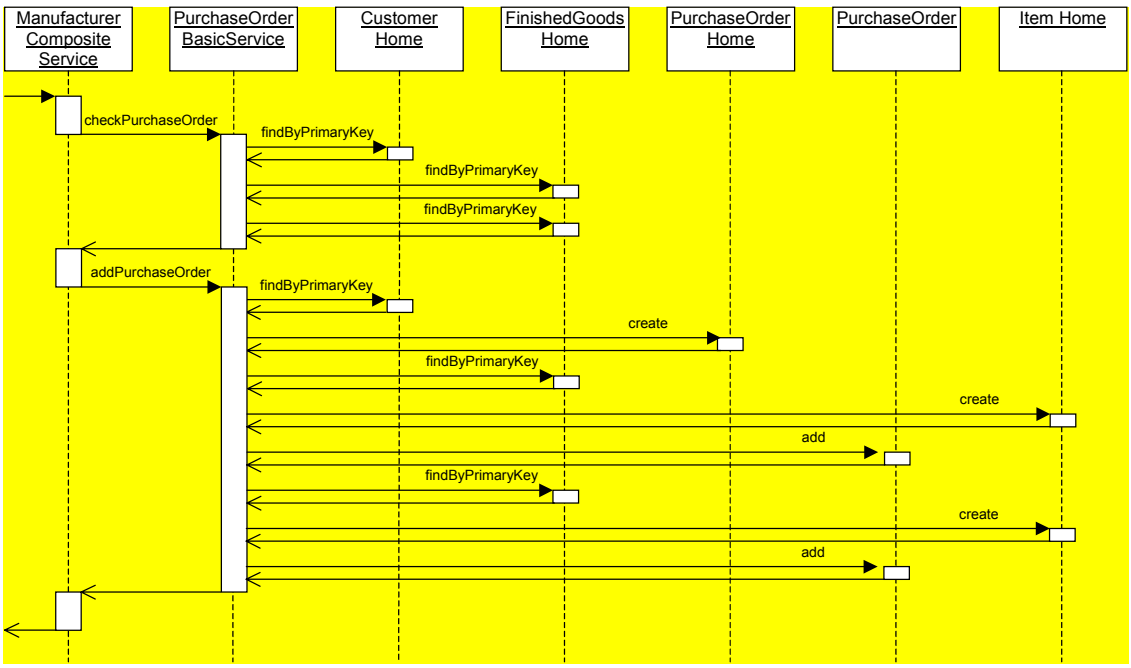


Figure 8: interaction diagram for submit purchase order invocation on the manufacturer.

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

Product Finished Goods Events

Production Lines: Black Ink Production Line	
Black Ink	0 Submit

Production Lines: Colored Ink Production Line	
Blue Ink	0 Submit
Red Ink	0 Submit
Yellow Ink	0 Submit

Production Lines: Paper Production Line	
Pink Paper	0 Submit
White Paper	0 Submit

Figure 9: expected results of step 4.

Manufacture 100000 units of "Black Ink" by entering 100000 into the input field, then press "Submit" This should be successful, and produce the following results:

Product Finished Goods	
Production Line	Black Ink Production Line
Finished Goods	Black Ink
Amount	100000
Status	Successful

Figure 10

Each press of this button has caused a servlet in the "Demo Web Pages" process to make a web service invocation to the manufacture composite service. This invocation is implemented by a JOpera web service, which makes two further web-service invocations this time on the basic service primary instance.

Step 5) Press the "Purchase Order Check" this button will show a web page, which shows the outstanding orders.

Purchase Order Check Events	

Purchase Orders	
MON1	
Customer Reference	TagAnz
0	e100000033a876d939909000
Total	100.0
Check	

Figure 11

Pressing on the "Check" button, this will see if the order can now be satisfied, and should result in the message "Now Complete", this means that an invocation has been sent to retailed containing shipment details (this cause the inventory of the retailer warehouse to be updated).

Pressing the button has caused a servlet in the "Demo Web Pages" process to make a web service invocation to the manufacturer composite service. This invocation is implemented by a JOpera web-service, which makes further web-service invocations this time on the basic service primary instance. If the purchase order can be satisfied ta web-service invocation is made to a retailer composite services, which the basic services to update the retailers inventory, to reflect a shipment.

Step 6) Press the "Submit Purchase Order" button, and Order 1000 units of "Black Ink, 500l Black Ink" by entering 1000 into the input field, then press "Submit".

Retailer Order Event: RON1	
Retailer Purchase Order Number: RON1	Status: Acknowledged

The result should be "Acknowledged", which indicates that there is sufficient stock at the retailer to satisfy the order.

Each press of this button has caused a servlet in the "Demo Web Pages" process to make a web service invocation to the retailer composite service. This invocation is implemented by a JOpera web service, which makes further web-service invocations this time on the basic service primary instance.

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.

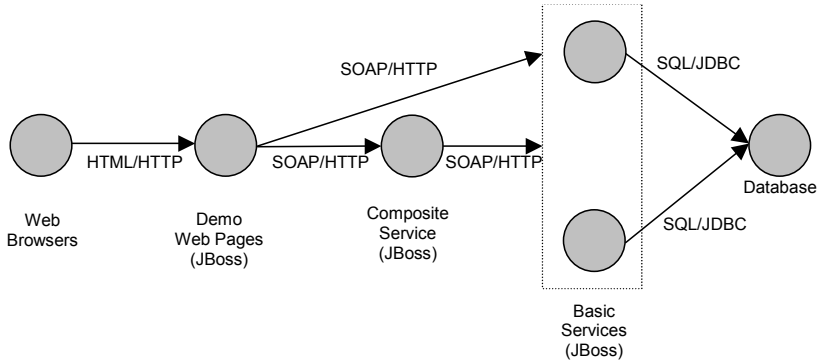


Figure 12: interactions within the demonstrator.

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 *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 are executed by JOpera as a workflow driven process. The interaction between the *Demo Web Pages* and the *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 and 14.

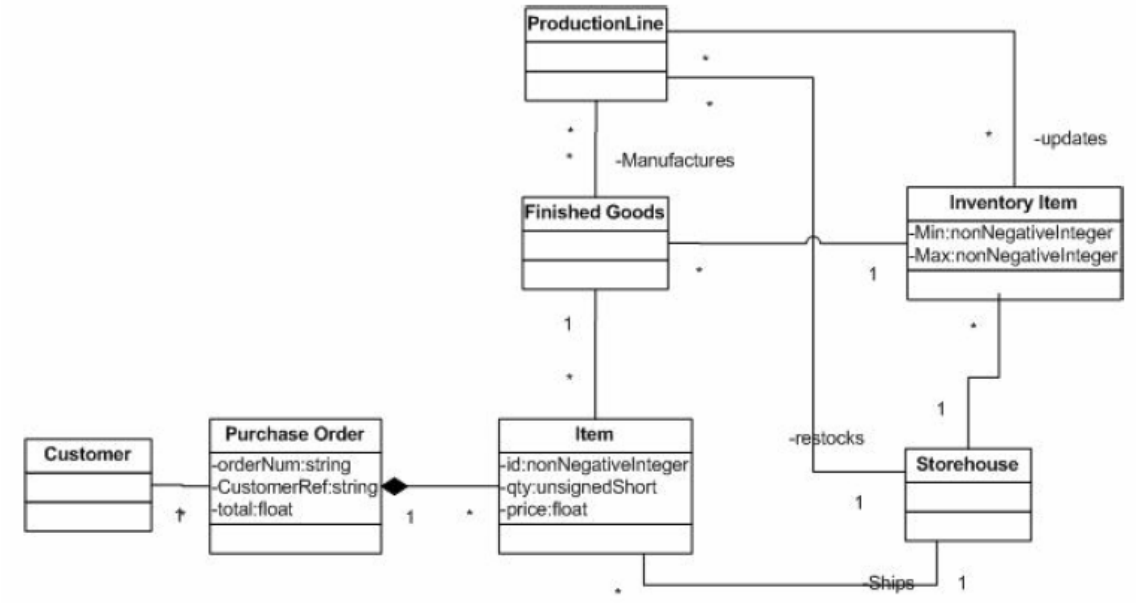


Figure 13: WS-I Sample Application Manufacturer persistent data model.

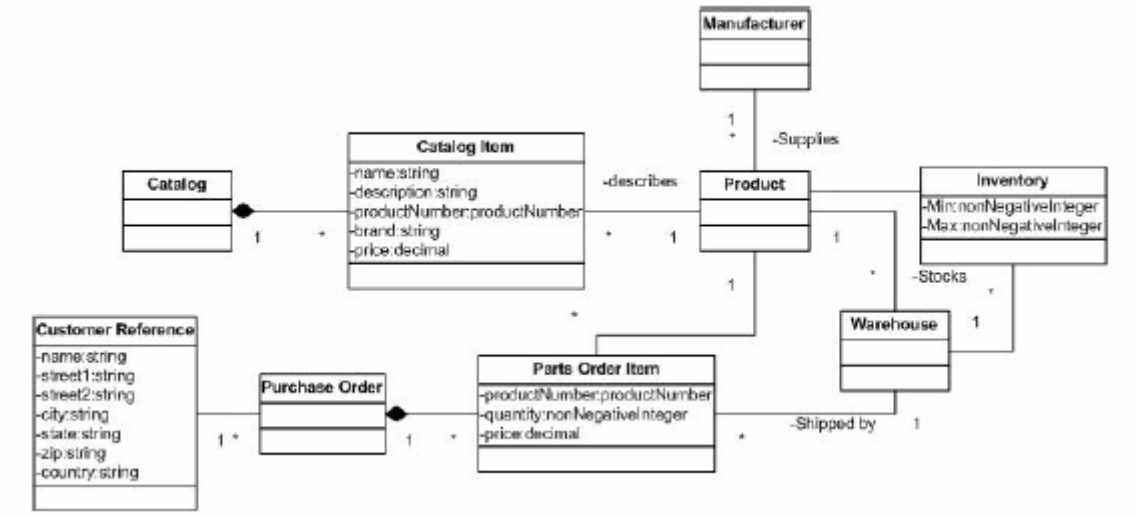


Figure 14: WS-I Sample Application Retailer persistent data model.

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 15 and 16.

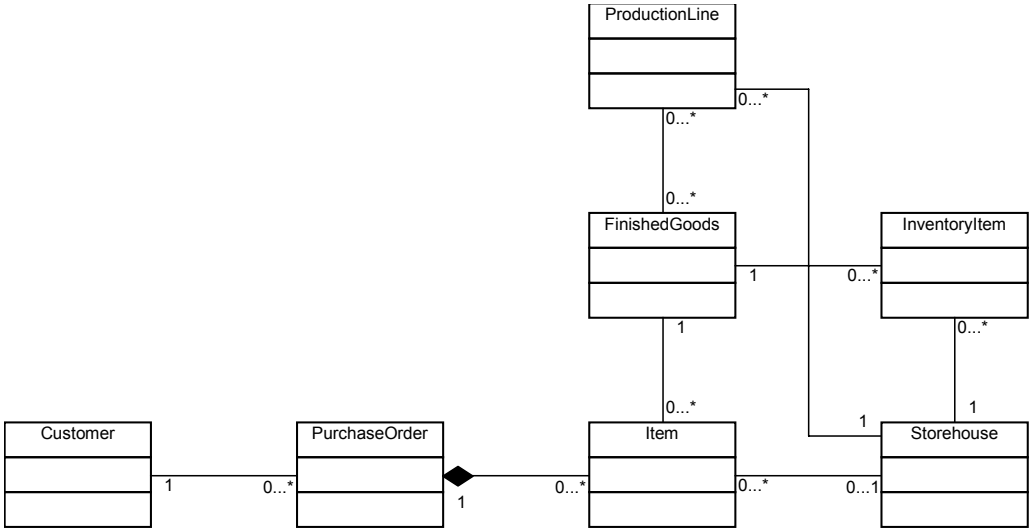


Figure 15: Manufacture System persistent data model.

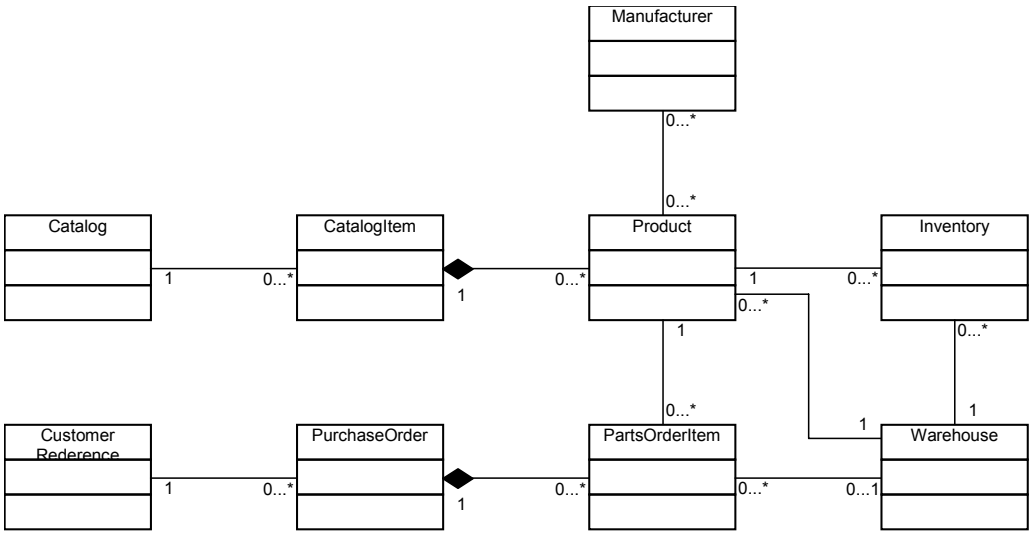


Figure 16: Retailer System persistent data model.

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)