Diploma Thesis

e-commerce applications:
Usage of Enterprise JavaBeans
in SUN's Petstore and IBM's Trade application

Thomas Wermter


Supervisors:
Dr John Murphy  Dr Thomas Seewaldt
Dublin City University, Ireland  Fachhochschule Mannheim, Germany
murphyj@eeng.dcu.ie  seewaldt@fh-mannheim.de
Foreword

This thesis is a result of my visit at the Performance Engineering Laboratory at the Dublin City University in Dublin, Ireland. I would like to thank my supervisor in Germany, Prof. Thomas Seewaldt for accepting an external work and Prof. Karl-Heinz Krauß who keeps up the contact with the engineering department at DCU. I also would like to thank Prof. John Murphy for inviting me, helping me organizing my visit and his supplying support. Thanks to the staff and members of the Performance Engineering Laboratory for useful tips and discussions.

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig, ohne Hilfe Dritter und nur mit den angebenen Quellen und Hilfsmitteln angefertigt wurde.
(I declare that this thesis is substantially my own work. Where reference has been made to the work of others this is acknowledged in the bibliography.)

Dublin, 24. February 2003

Thomas Wermter
Abstract

The performance of component-based applications is very important, especially in large-scale systems. Petstore is a component-based sample application by SUN, in which performance measurements are supposed to be accomplished automatically. Petstore has over 30 Enterprise JavaBeans (15 in the essential Petstore web shop), and therefore it should be a good start to test the quality and results of the measurement-framework (COMPAS). At this quite small application, in contrast to big applications with hundreds of components, it is also possible to have a look at the interconnections of the components by hand. Therefore, it will be useful to get different UML-diagrams (sequence diagrams) out of the source code of Petstore and see, by run-time analysis, which customer actions on the web-interface causes which actions in the business layer (EJB tier). With this information it should be possible to gain knowledge of possible bottlenecks or performance problems in the Petstore application, which can then be compared with the results of the COMPAS framework.

Furthermore, Trade 2/3 are benchmark applications by IBM for performance tests with the Websphere application server. These applications also use SUN's J2EE technology and shall be integrated in COMPAS as well. To be able to make statements of the quality of the results measured by COMPAS, it is necessary to have comparative values about the performance, preferable on different platforms.
Table of Contents

Foreword....................................................................................................................................................... II
Abstract............................................................................................................................................................ III
Abbreviations................................................................................................................................................ VI
1 Introduction................................................................................................................................................... 1
2 J2EE Technology........................................................................................................................................... 2
  2.1 Servlets................................................................................................................................................ 3
  2.2 JavaServer Pages.................................................................................................................................... 3
  2.3 Enterprise JavaBeans............................................................................................................................ 4
    2.3.1 Entity beans................................................................................................................................... 5
    2.3.2 Session Beans................................................................................................................................ 5
    2.3.3 Message Driven Beans............................................................................................................... 6
  2.4 J2EE Patterns....................................................................................................................................... 7
    2.4.1 Front Controller......................................................................................................................... 7
    2.4.2 Intercepting Filter....................................................................................................................... 8
    2.4.3 Session Facade........................................................................................................................... 9
    2.4.4 Data Access Object (DAO)....................................................................................................... 9
3 Performance measurements......................................................................................................................... 11
  3.1 OpenSTA.......................................................................................................................................... 11
  3.2 IBM Sample Application Trade 2..................................................................................................... 12
    3.2.1 Actions....................................................................................................................................... 13
    3.2.2 Configuration options............................................................................................................. 13
  3.3 Trade 3............................................................................................................................................. 14
    3.3.1 Architecture........................................................................................................................... 14
    3.3.2 Configuration options............................................................................................................. 15
  3.4 Performance benchmark..................................................................................................................... 16
    3.4.1 Environment........................................................................................................................... 16
    3.4.2 Configuration options............................................................................................................. 16
    3.4.3 Workload generation.............................................................................................................. 17
    3.4.4 Benchmark execution............................................................................................................. 19
4 Petstore...................................................................................................................................................... 21
  4.1 Architecture....................................................................................................................................... 21
    4.1.1 EJBs in Petstore.................................................................................................................... 22
Abbreviations:

AOP Aspect Oriented Programming
API Application Programming Interface
AWT Abstract Window Toolkit
BMP Bean Managed Persistence
CGI Common Gateway Interface
CMP Container Managed Persistence
COMPAS Component Performance Assurance Solutions
EAR Enterprise Archive
EIS Enterprise Information System
EJB Enterprise JavaBeans
IDE Integrated Development Environment
GUI Graphical User Interface
HTML Hypertext Markup Language
IIOP Internet Inter-ORB Protocol
J2SE Java 2 Standard Edition
J2EE Java 2 Enterprise Edition
JAF JavaBeans Activation Framework
JAXP Java API for XML Processing
JAXR Java API for XML Registries
JAX-RPC Java API for XML based RPC
JDBC Java Database Connectivity
JMS Java Message Service
JNDI Java Naming and Directory Service
JSP Java Server Pages
JTA Java Transaction API
JVM Java Virtual Machine
LDAP Lightweight Directory Access Protocol
MVC Model-View-Controller
ORB Object Request Broker
RDBMS Relational Database Management System
RMI Remote Method Invocation
RPC Remote Procedure Call
SDK Software Development Kit
SNMP Simple Network Management Protocol
UML Unified Modeling Language
URL Uniform Resource Locator
VAJ Visual Age for Java
WAF Web Application Framework
XML Extensible Markup Language
1 Introduction

The nature of the task for this work is to get material for a comparison with automated results produced by a framework named COMPAS (Component Performance Assurance Solution). This generic framework can monitor and model any J2EE compliant distributed application. With the gathered data the framework can also make performance predictions. For the result comparison two applications were chosen. First, Petstore which is a sample application from SUN, and secondly, Trade 2 or 3, an application from IBM for performance benchmark tests.

Sequence diagrams shall be extracted from Petstore which can be compared to the models COMPAS creates. These diagrams are needed especially for the business logic of the store. Here, the Enterprise JavaBeans are located and can easily develop to the performance bottleneck of a system.

The Trade application already comes with some UML diagrams. Here, performance benchmarks on different operating systems are needed to have data for comparison when the same benchmarks are performed with the COMPAS framework monitoring the application.

Chapter 2 explains the basics of the J2EE technology and those design pattern which are essential for the understanding of this thesis. The examples for the design pattern are already taken from the Petstore application. That way, it is easier to recognize and understand the pattern in the later described scenarios. This results in a pattern description in which not every detail is discussed because of a simplified scenario. Yet, the general idea should be comprehensible and for further information [SUN02b] can be referred to.

Chapter 3 is about the performance measurements in Trade 2 respective Trade 3. Configuration options and workload generation are described as well as the steps performed for the benchmark tests.

Chapter 4 shows the architecture and the way how to get logging information of the Petstore application. Here, also the business logic is described in different scenarios and sequence diagrams.
2 J2EE Technology

J2EE (Java 2 Enterprise Edition) is based on the standard edition (J2SE) and tries to simplify and improve the developing of distributed software development. J2EE is not a product you can buy but a collection of components, technologies and standards companies have to follow and fulfil if they want to develop J2EE compliant server or applications.

J2EE addresses different problems [SIN02]:

• Simplified architecture and development
• Freedom of choice in servers, tools and components
• Integration with existing information systems
• Scalability to meet demand variations
• Flexible security model

Figure 1 shows the J2EE architecture with the four application components. Two of them, Applet Container and Application Client Container, are settled on the client-side. Applet Container, usually executed in web browsers, can provide a powerful GUI without the need of installing or updating because applets are loaded from the Web Container. As depicted, communication is limited to these two components. Applets are not able to deal with a database or an EJB Container directly. If such a behaviour is needed a standalone Java program (component Application Client Container) is the only choice.
On the server-side the Web Container holds the Servlet and JavaServer Pages technology. It is able to respond to HTTP requests and can produce HTML or binary answers. As required by the J2EE specification, the Web Container and other J2EE protocols have to be included in a web server. The EJB Container provides support for transaction oriented business logic in applications. The server-side technologies will be discussed in the following sub chapters. The Enterprise Information System (EIS) tier can include databases or legacy servers. Here, normally the persistent data is stored.

### 2.1 Servlets

Servlets are server-side Java programs which gain the advantages from the Java run-time environment, like lightweight threads for each instance in contrast to more time consuming processes. Servlets are SUN’s portable answer to the Common Gateway Interface (CGI) technology. In [HAL01] servlets are described as follows:

"Java servlets are more efficient, easier to use, more powerful, more portable, safer and cheaper than traditional CGI and many alternative CGI-like technologies."

Servlets can be used for read data sent by users, not necessarily only from web forms. They can generate results dynamically and send them back to clients, HTTP or even binary answers. For small applications it is also common to include the business logic in Servlets to avoid the overhead an additional layer with an EJB container comes with. Therefore, it is possible to access information from the EIS tier directly.

### 2.2 JavaServer Pages

JavaServer Pages (JSP) is a technology to combine static and dynamic content in one HTML page, in which the dynamic information can be generated by servlets. Therefore, new HTML-like tags can be defined and then can be used by a webpage designer without programming skills or any knowledge about the implementation of those special tags. This way JSP helps to allocate tasks to the particular specialists, like programmer or webpage designer as mentioned before. It is also useful to use JSP to separate content from its presentation.

Actually, every problem could be solved with servlets only. But, depending on the application it can be more convenient to change the HTML page instead of using a lot of println statements in a servlet, because it is not a good coding style since it makes maintenance more complicated. In fact, JSP documents are translated into servlets in the web container.
2.3 Enterprise JavaBeans

Enterprise JavaBeans (EJB) is a component architecture developed for the business logic part in the middleware layer. Developers can fully concentrate on implementing the business logic because the so-called *EJB container* provides a run time environment and certain services for enterprise beans. The EJB container has to take care of creating and deleting instances of beans, as well as making the availability of beans transparent to a client. For performance reasons the container can support a pooling of bean instances, which means to keep a certain amount of unused beans until a client sends a request. With this technique it is possible to omit the overhead of creating and deleting for every request. In detail, the EJB specification 2.0 [SUN01] particularizes the following services an EJB container must support:

- Java 2 Platform, Standard Edition v1.3 (J2SE) APIs
- EJB 2.0 APIs
- JNDI 1.2 (Java Naming and Directory Interface)
- JTA 1.0.1 (UserTransaction interface only)
- JDBC 2.0 extension (Java Database Connectivity)
- JMS 1.0.2 (Java Message Service)
- JavaMail 1.1, sending mail only
- JAXP 1.0 (Java XML Processing).

The final draft of EJB specification 2.1 [SUN02] includes a few more services and newer versions of the ones listed above, basically to add web service functionality to the Enterprise JavaBeans architecture (stateless session and message-driven beans will be accessible by SOAP-based web services).

In difference to JavaBeans, which is also a component architecture of SUN, the EJB model is focusing on distributed, server side and transaction-oriented components. To make EJBs accessible to a client on a different machine, the developer can implement Remote and Home interfaces (remote client view). The communication between the two partners is then based on RMI-IIOP (Remote Method Invocation over Internet Inter-ORB Protocol). If the client resists in the same JVM (Java Virtual Machine) there is no need for such an overhead remote protocol. Therefore, the developer can implement Local and LocalHome interfaces (local client view). In this case, the transparency of the bean's location is lost but the performance increased. It is possible to implement the remote and the local view in one bean, but the specification advises to decide on one of them. The significance of considering the specification is the portability of an component between different EJB container vendors.
The structure, type and dependencies of an EJB are described in a deployment descriptor. This file is written in XML format and can combine several beans. The EJB container gets information how to deal with the components during run time from this file.

There are three different types of enterprise beans: entity beans, session beans and message driven beans. The characteristics of each will be explained in the following paragraphs.

### 2.3.1 Entity beans

Entity beans usually represent persistent data from 'real-world objects', for instance information about a customer, bank account or a certain product from a database. Several clients can access an entity bean simultaneously. The EJB container has to make this access available and thus the developer does not have to bother with it. To a client it looks like it has exclusive access to the bean.

One or more attributes have to be chosen as a primary key to unambiguously identify an instance of an entity bean. The combination of the primary key and EJBHome object is known as entity bean identity.

Entity beans can be subdivided into two persistence concepts:

1. **Container Managed Persistence (CMP)**
   - The EJB container is responsible to take care of the persistence. The developer only has to create abstract getter and setter methods for attributes. During deployment the container will implement the methods in an inherited class.

2. **Bean Managed Persistence (BMP)**
   - The developer has total control over the persistence of bean attributes.

### 2.3.2 Session Beans

Session beans represent the business logic of an application, for example the transaction from one bank account to another. Different than entity beans a client has exclusive access to one session bean at a time. But the session beans are transient (or not persistent), and if their changed state should be kept for more than the current session it must use entity beans or access a database directly. There are two kinds of session beans:

1. **Stateful Session Beans**
   - The state of a session bean is known as conversational state. If a session bean's state can be changed by calling its methods, the component is called stateful session bean and it has a conversational state. The bean is then providing create methods, which expects parameters for initialising the state. A conversational state also means that every client has to have its own bean for the whole session.
This would lead to an unlimited amount of beans during parallel requests from many clients. For that reason the beans provide a *passivate* method to move the component out of the main memory. For restoring the bean an *activate* method is provided.

2. Stateless Session Beans

A *stateless session bean* on the other hand does not have a state which can be altered, therefore it always has a *create* method without parameters. A client has only exclusive access during one method call, because the bean is stateless and no method call can change the state of the bean. The next method call can be processed by a different session bean.

2.3.3 Message Driven Beans

A client cannot access a message driven bean directly. The communication is carried out through a messaging service provided by the EJB container, and in J2EE it is called JMS (Java Messaging Service). The client sends a request to the messaging service and from there the message is send to the message driven bean. The client stays totally anonymous to the bean and if the bean needs to identify a client, a unique identifier needs to be send together with the message.

The main feature of JMS is the asynchronous communication. The client does not need to wait for the server to return after a remote method call. After the message is sent the client can continue executing the program code. This is useful for decoupling long ongoing processes, for example orders which must be confirmed by hand.

In messaging systems two different communication concepts exist:

1. **Point to Point**

   The message is sent from one sender and is delivered over the messaging system to exactly one receiver. The sender addresses messages to a specific *queue* and the receiver has to send back an acknowledge after processing the message successful.

2. **Publish and Subscribe**

   Here a message can be received from more than one client. In fact, receivers subscribe to a *topic* and get from now on all published messages from a sender.


2.4 J2EE Patterns

In general, Software Patterns provide developers with practical solutions for reoccurring problems during the software developing process. Usually, patterns are developed by software architects with a great experience because they repeatedly see similar problems and describe a general re-usable solution for this kind of question. A detailed description, in the famous book by the “Gang Of Four” [GAM95] because of the four authors, patterns became very popular, especially in object-oriented languages.

SUN’s J2EE resulted in the arising of new problems and the creation of new or adapted patterns for this new technology [ALU01]. Here, only those patterns will be discussed which are necessary for the understanding of the following description of the sample application. Furthermore, common patterns, like Model-View-Controller (MVC) or Factory, will only be briefly mentioned and described where it is necessary, as they should be well-known to developers and can be found in [GAM95] or [GRA98] and are not essential for the overall understanding.

Illustrations made in the following sub-chapters are already related to the Petstore application and contain class names used in it. This results in a better recognition in the later discussed business scenarios. If available, illustrations were taken from [SUN02b].

2.4.1 Front Controller

Client requests need to be processed on the server side, but how do the requests come to the proper handler routine on the server? One possibility is to provide the client with knowledge to which server component it has to address a certain request. That means, a client needs information about the server implementation which in turn will result in absolutely necessary client changes when the server design alters.

![Figure 2 sequence diagram: Front Controller [SUN02d]](image-url)
A better solution is to realize one central place on the server where the client can send its request to. This central location, the Front Controller, takes care of the following processing of the request and will send a response back to the client. Now, a change of the server request handlers results only in a change of the Front Controller.

As seen in Figure 2, the client sends its request to the server (transition number 1) where the Front Controller, here named MainServlet, processes the request (with supporting Helper classes). After dispatching the request for further processing (2) to the application Model (business logic) the Front Controller sends the request to a FlowManager (3), which decides what the next visible view for the user should be. Finally, the controller passes the request to the ViewAssembler to build a webpage based on the previous selected view and a webpage template (5). The page can now be presented to the client.

2.4.2 Intercepting Filter

This pattern is used for requirements that occur for all client requests such as logging, auditing or security. Instead of providing every component with method calls to a security service to see if a user is logged in the Intercepting Filter pattern, for example, is a way to pre- or post process client requests if a certain requirement is met. To remain with the security example, such a requirement could be a protected webpage which only is accessible after the user is signed-on. The filter seizes this request and checks if the user is already logged in, otherwise a login webpage is presented.

Figure 3 shows the example of the sign-on process in Petstore (a pre-processing filter). As long as a client requests unprotected webpages inquiries will be directly handled by the MainServlet (1) as described in Chapter 2.4.1. When a protected resource is requested the SignOnFilter intercepts (2) and checks if the user is already signed on, otherwise the user will be asked for the login information (username and password). Provided that this information is correct the user will be logged in by a stateless session bean, named SignOnEJB (3), and the original protected request will now be forwarded to the MainServlet (4).
Figure 3 is a simplified sequence diagram for the sign-on event with only one preprocessing filter. Filters are called by a FilterChain which in turn is created by a FilterManager. The FilterChain is able to delegate requests to several independent filters.

### 2.4.3 Session Facade

The Session Facade is an adapted Facade pattern. Its goal is to simplify the access of a variety of business objects in which interactions can grow complicated. Programmers, who want to use the business logic, do not need (and want) to know about the difficult interconnections of the beans. A Session Facade is hiding this structure in a few methods, which are easier to use. As the name already indicates, it is realized as a session bean, either stateful or stateless. If it is necessary to store information in the Session Facade a conversational state is needed and, therefore, the stateful session bean must be chosen.

A side effect is the decoupling of components. For that reason, method calls between those “hidden” components are minimized and the architecture is easier to understand. This makes the application's design more flexible.

![ShoppingClientFacade](image)

**Figure 4 Session Facade**

Figure 4 shows the ShoppingClientFacade used in Petstore. It hides all methods related to a customer and takes care of creating and looking up the objects respectively. For components which have to use these methods it is a great simplification.

### 2.4.4 Data Access Object (DAO)

Almost every application needs to access data from a persistent data source, like, for example, a RDBMS, LDAP repositories or in the easiest case a simple text file. Access to such sources is, to a large extend, dependent on APIs from the storage vendors. To keep business logic code free from dependent API calls the Data Access Object pattern is the first choice. It ties all calls to a data
source together and is therefore the only part which needs to be altered if the data source has been changed. This is guaranteed because a DAO contains not only the logic to access the EIS tier, it also manages the connection to the source.

In Figure 5 this pattern can be seen in combination with the Abstract Factory pattern. This combination is powerful and makes switching to other data sources easy. The CatalogDAOFactory decides during run-time which CatalogDAO needs to be created. This information is picked up from a XML deployment descriptor file.

Figure 5 Data Access Object pattern
3 Performance measurements

Performance in J2EE applications depends on various things. First of all on the main application with J2EE components (EJB), but there are also application servers of different vendors, and the diversity of operating systems can affect the performance, for example the different implementation of threads.

A generic framework, independent from the platform or application server that monitors the J2EE application in real time, gains information about the hot spots. This happens in a non-intrusive way, therefore, there is no need for developers to have access to the source code, nor to change the components.

3.1 OpenSTA

OpenSTA is an open source testing architecture for distributed applications. It produces workload and monitors web based applications. The monitored data is visualized in graphs and a export feature makes further processing and analysis easy. Monitoring includes information, like response time on client and hardware resource utilization on the server-side. OpenSTA offers three ways to control a workload test:

• Collectors
• Scripts
• Tests

Collectors are used to monitor performance of a computer. It gains the data either from the performance monitor which comes with MS Windows operating systems, or from SNMP agents in a network. The Simple Network Management Protocol is used to manage and monitor any network device or computer which provides an SNMP agent.

Scripts are used to produce the actual workload for the target application. OpenSTA comes with its own scripting language to make the communication between the clients and the server flexible. Also, a recording feature is provided. The Script Modeler intercepts the traffic between a web browser and server-side application produced by a user. A Script, generated like this, is the starting point for manual changes to adapt it to the speciality of an application, like login several users with different usernames.

Tests are the place where Collectors and Scripts meet. Different Scripts can be combined in two different ways: Tasks and Task Groups. A Task Group can contain several Tasks and an amount of Virtual Users. A Task Group is always executed by a particular host and can be configured for
different start and termination times, how actions of Virtual Users are distributed over time and the number of Virtual Users which should be monitored for the results. A Task on the other hand, is the execution of exactly one Script. A Task can run for a fixed time or a number of times.

As mentioned before, workload is generated by Virtual Users on local or remote machines. The scripting language and the automated recording of user actions was used to build Scripts which reproduced different user behaviour. For benchmarking the Trade application, three scripts were generated and customized. These can be found in the appendix. One is for quote operations, the other for stock buys and the last one for stock sells.

3.2 IBM Sample Application Trade 2

IBM published a sample application named *Trade 2* [IBM01] for their application server WebSphere. It is a possible alternative to check the produced results from the COMPAS framework. In difference to SUN's Petstore application, which only shows as much as possible from the J2EE technologies, IBM produced a configurable benchmark application. Trade 2 is modeled after an online brokerage firm and uses servlets, JSP, EJB and JDBC, very like the SUN's sample to provide web based services. Altogether Trade 2 has 9 Enterprise JavaBeans: 2 Session and 7 Entity beans. IBM provides three ways to run performance benchmarks:

1. **TradeAppServlet**
   
   In this way a user would normally interact with the web interface. Users can log in by filling out a HTML form and buy or sell stocks. For an automated benchmark the work load generator has to be able to fill out HTML forms and post dynamic data.

2. **TradeScenarioServlet**

   For each call of the URL this servlet chooses a random user and trade operation. Every reload a new requests is generated. Like this, the web load generator only has to reload the same URL again and again.

3. **Web Primitives**

   The web primitives can produce workload for certain components of the WebSphere J2EE technology. This way it is possible to focus testing on certain components, like EJBs, the servlet engine, JDBC, or JSP. But also a whole path from a servlet to a database request through a session and entity bean can be traversed. The web primitives are also provided with source code therefore they can easily be integrated in self-developed benchmarks.
3.2.1 Actions

Actions are high level user operations in Trade 2 application. There are 10 different actions implemented which can be seen in Table 1. A description of the actions and the resulting database activity can be found in the installed Trade 2 documentation.

The actions can be organized into groups which make use of different parts of the application:

- actions Home/Quote: Servlet and JSP
- actions Login/Logout/Register: Session and database updates
- actions Account/Portfolio: database read
- actions Buy/Sell/Account Update: complex database update

3.2.2 Configuration options

Trade 2 comes with different options which can be set in a web configuration interface. One option is the run-time mode. The user has the choice to use mode “EJB” in which database access is made through EJBs which in turn are wrapped for performance reasons by a special kind of IBM beans, named IBM VAJ EJB Access beans. These beans can be turned off by using the mode “EJB_ALT”. The last mode, “JDBC”, does the benchmark completely without using EJBs.

Another option to configure is the scenario workload mix. The user can decide if there should be more or less database operations, an equal distribution of every action or if no new user registrations should be performed. IBM provides five workload mixes:

- Standard
  Referring to the documentation the standard mix supplies a mix of operations “not unlike customer environments”.

- DB-Light
  Less database operations are performed than in the standard mix

- DB-Heavy
  Actions are stressed which result in a heavier database operation (buy/sell)

- Uniform
  Every action is equally executed.

- No register
  The same as the standard mix but without new user registrations.
Table 1 depicts the actions and the workload mixes. Furthermore, the actions are grouped as mentioned in chapter 3.2.1. Every action has a certain percentage assigned (first column in every workload mix). The addition of those percentages for every action group is shown in the second column in every workload mix. Login has to be zero because the Logout action represents both, and each logout results in a new login when a new action for a user is performed.

The percentages cannot be altered in the web interface. If user defined values are needed, they must be changed in the source code of the class file TradeConfig.java.

<table>
<thead>
<tr>
<th>Action</th>
<th>Standard</th>
<th>DB-Light</th>
<th>DBHeavy</th>
<th>Uniform</th>
<th>No register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Logout</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>Register</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Home</td>
<td>20</td>
<td>60</td>
<td>75</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>Quote</td>
<td>40</td>
<td>50</td>
<td>30</td>
<td>11</td>
<td>41</td>
</tr>
<tr>
<td>Account</td>
<td>7</td>
<td>15</td>
<td>10</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>Portfolio</td>
<td>8</td>
<td>5</td>
<td>12</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>Account Update</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Buy</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>Sell</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>11</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 1 Trade 2: Actions and distribution on workload mixes (numbers are percentages)

3.3 Trade 3

Trade 3 is the extended sequel of the Trade 2 application. Its architecture is very similar to the predecessor Trade 2. Trade 3 is compliant to the J2EE 1.3 and EJB 2.0 standard. Like these standards, the application’s features increased as well. Only changes and differences will be discussed in this chapter. Features which stayed the same and mentioned before are not worked out.

3.3.1 Architecture

The client communicates to the server-side web container. Either with a web client to a servlet or with a SOAP client to WebSphere’s SOAP Router. A third option will be to make calls to web services but this feature is not yet implemented. The web container sends requests to a dispatcher session bean TradeSessionEJB in the EJB container. This bean takes care of the further processing of the request, accesses entity beans and sends, if asynchronous order-processing is enabled, messages to the message server.

Altogether, Trade 3 has eight enterprise beans. Six entity beans, two session and two message driven beans:
• Entity beans:
  KeyGenBean
  AccountBean
  AccountProfileBean
  QuoteBean
  OrderBean
  HoldingBean
• Session beans:
  TradeBean
  KeySequenceBean
• Message driven beans:
  TradeBroker
  TradeStreamer

3.3.2 Configuration options

Trade 3 comes with two different workload mixes only, standard and heavy. Table 2 shows the
distribution between the ten actions.

<table>
<thead>
<tr>
<th>Action</th>
<th>Standard</th>
<th>Heavy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Logout</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Register</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Home</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Quote</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Account</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>Portfolio</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>Account Update</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Buy</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Sell</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 2 Trade 3: Actions and distribution on workload mixes (numbers are percentages)

It is possible to avoid the usage of EJBs at all. Access to the database and messaging service is then
made through direct JDBC and JMS code. In addition, order processing can be altered to execute
tasks either synchronous or asynchronous. The latter uses the TradeBrokerMDB for the processing
of orders. The communication between the servlets from the web container and the business logic
can take place through Java RMI, which is the default, web services and SOAP RPC.
3.4 Performance benchmark

To see the impact of the COMPAS framework on performance, a benchmark test with the Trade 3 application is performed. Therefore, a reference benchmark has to be carried out to be able to recognize changes when used with COMPAS. The following sub chapters provide the necessary information about configuration of hard- and software.

As it is the beginning of the benchmark tests only the basic actions should be included in the benchmark. But the workload should still be based on an usual user behaviour, thus, sessions of buying, selling and quoting were recorded as well as user think times between the certain actions.

3.4.1 Environment

For the benchmark, the hard- and software as shown in Table 3 has been used.

<table>
<thead>
<tr>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 2000 Advanced Server SP3</td>
<td>IBM NetFinity RS6000R server(4-processor)</td>
</tr>
<tr>
<td>IBM Websphere Application Server 5.0</td>
<td>Adaptec AIC-7899 Ultra160/m PCI SCSI</td>
</tr>
<tr>
<td>IBM DB2 7.2 Fixpack 7</td>
<td>IBM NetFinity Fault Tolerance network adapter</td>
</tr>
<tr>
<td>OpenSTA 1.4.1</td>
<td>512 MB RAM</td>
</tr>
<tr>
<td>IBM's Trade 3 benchmark application</td>
<td>34 GB IBM ServeRAID SCSI</td>
</tr>
</tbody>
</table>

*Table 3 Hard- and Software equipment for the benchmark.*

3.4.2 Configuration options

To be able to reconstruct the benchmark exactly, the chosen configuration options of Trade 3 are summarized in tabular form in Table 4.

<table>
<thead>
<tr>
<th>Option</th>
<th>Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-Time Mode</td>
<td>EJB</td>
</tr>
<tr>
<td>Order-Processing Mode</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Access Mode</td>
<td>Standard</td>
</tr>
<tr>
<td>Scenario Workload Mix</td>
<td>Standard</td>
</tr>
<tr>
<td>WebInterface</td>
<td>JSP</td>
</tr>
<tr>
<td>Trade Max Users</td>
<td>2000</td>
</tr>
<tr>
<td>Trade Max Quotes</td>
<td>1000</td>
</tr>
</tbody>
</table>

*Table 4 Trade 3, selected configuration*
3.4.3 Workload generation

OpenSTA was used to produce workload for the Trade 3 application. The scenario for the first benchmarks is:

• 60% of users performing stock quotes
• 20% buy stock
• 20% sell stock

Therefore, three workload scripts are recorded and adapted. The 2000 user accounts are split on these scripts. Consequently, every script contains 666 and 667 users respectively. Table 5 shows the common features of the scripts and the following paragraphs explain the characteristics of each in detail. For each script two virtual user think times are defined. Scenario Time I contains the recorded, but rounded, think times, whereas Scenario Time II contains heavily reduced think times for producing a lot of concurrent workload. This scenario was mainly used to find out the saturation point.

<table>
<thead>
<tr>
<th></th>
<th>Quote</th>
<th>Buy</th>
<th>Sell</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP GET operations</td>
<td>13</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>HTTP POST operations</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Scenario Time I (s)</td>
<td>71</td>
<td>77</td>
<td>57</td>
</tr>
<tr>
<td>Scenario Time II (s)</td>
<td>7,6</td>
<td>9,6</td>
<td>6,3</td>
</tr>
</tbody>
</table>

Table 5 scenario comparison

To achieve the percentages mentioned above three quote scripts, one buy and one sell scripts are combined. This means, that there are 5 tasks altogether which can have some more options assigned. Every task gets the same number of virtual users. A scenario is performed twenty times by each user and the pause between such an iteration is 5 seconds. For the duration while one user is active the following formula can be used:

\[
t_{ua} = (\text{script duration} \times \text{number of iterations}) + (\text{pause between iterations} \times \text{number of iterations})
\]

In a task, the users are not starting their activity altogether simultaneous. To avoid that too many users start their requests at exactly the same time, different intervals between user introductions are chosen. But this is only necessary for the three quote scripts because in the other scripts users have different think times. That is enough to distribute the requests. The following times \( t_{ui} \) have been picked:

• Quote scripts: 13s / 15s / 17s
• Buy script: 15s
• Sell script: 15 s
With these information the duration of a benchmark run can be determined:

\[ t_b = \text{longest } t_{ua} + \text{longest } t_{ui} \times \text{number of VU} \]

Here, this means the duration of the buy script has to be used and we get an value of

\[ t_{ua} = 1640s \]. The longest time between two user introductions is in the third quote script (17s).

For an assumed virtual user number of 15, the duration of a benchmark run is \( t_b = 1895s \) or about 31.5 minutes.

**Quote stock script**

The script for quoting stocks logs a virtual user in, performs stock quotes and logs the user out afterwards. The sequence looks like this:

1. Login
2. 3 times Quoting different stocks
3. Logout

**Buy stock script**

For buying stocks a virtual user logs in and quotes a randomly selected stock. After that, the user buys 100 of the quoted stock and makes a request for his portfolio before logging out. Again, the sequence:

1. Login
2. Quote a stock
3. Buy stock
4. Show portfolio
5. Logout

**Sell stock script**

This script is a special case because after the login of a virtual user it has to be checked if there is at least one stock in the user's portfolio. If there is one, then it can be sold. If no stock can be found the user is logged out immediately without further actions. The sequence for selling a stock:

1. Login
2. Show portfolio
3. Check, if a stock is available (if no, then logout)
4. Sell stock
5. Show portfolio
6. Logout
3.4.4 Benchmark execution

To get valid performance information it is necessary not to be limited by a hardware bottleneck. Once, the scripts proved to be working correctly, an increasing workload was used to identify this saturation point. Hence, different resources have been monitored:

- CPU
- harddisk
- network
- RAM

It showed that the harddisk utilization reached several 100% peaks if more than 65 virtual users made concurrent requests. As problem, the write access to the disk was clearly identified. Figure 6 and 7 show an example of 85 virtual users in total and the corresponding harddisk utilization.

Before every benchmark run the Reset Database command was executed. This link can be found on Trade’s web configuration page. The first few runs showed also that the script which sells stock was out of work because not every user account had stocks in its portfolio. For that reason, an additional script was written to fill the portfolio of users who are selling stocks. This task was also executed before starting the benchmark.
Figure 8 and 9 depict the users and harddisk utilization for the benchmark run with 65 virtual users. The harddisk activity peaks are almost always under 85% and the concurrent users sway between 58 and 62. The throughput for this run went up to 27 HTTP responses per second at its maximum and response time stayed mostly below 100ms. The graphs for throughput and response time are shown in Figure 10 and 11. The values for the other monitored resources did not point to bottlenecks or problems. The available free RAM did not fall below 120MB and the overall processor utilization is almost not worth mentioning as it stayed constantly under 10%. The network traffic with peaks at 75KByte/s for sending and 13KByte/s for receiving no problem for server and network.
4 Petstore

The e-commerce web application Petstore is a free sample J2EE application from SUN Microsystems. It is developed by SUN's Blueprints Team as a reference application for developers to learn how to write distributed "write once, run anywhere" enterprise applications. It illustrates the usage of key technologies included in the J2EE platform, like JavaServer Pages and Enterprise JavaBeans, and provides a framework in which design patterns, reusable components and design recommendations are extensively applied.

At the quite small Petstore application with 31 Enterprise JavaBeans, in contrast to big applications with hundreds of components, it is also possible to manually have a look at the interconnections of the components. Therefore, it is useful to get different UML sequence diagrams from the source code of Petstore and see, by run-time analysis, which customer actions on the web-interface causes which actions in the business layer (EJB tier). Log statements and debugging help to get the correct data and to gain the necessary information from the source code without jumping from method call to method call manually, which is a susceptible way to make errors.

4.1 Architecture

The main goal of the Petstore developers was to show as many technologies of the J2EE platform as possible ([SIN02], p.347). The architectural approaches should be kept in mind as a good design for applications, but performance was never an issue. Therefore many design patterns and different layers have been used.

The sample application is divided into two major functional units:

1. The web front end

Customers interact with the application through a web interface. They can browse through the pet database, add pets to a shopping cart and place orders. This front end is built after the Model-View-Controller (MVC) architecture, which is used to decouple the presentation layer (view) from the data layer (model). When a model is changed by a controller, it will notify all registered views to update their information. As a result, applications following the MVC architecture are easy to maintain and extend.
2. The Order Fulfillment Center (back end)

The Order Fulfillment Center processes customer orders. The fulfillment section contains the business and transaction logic. Incoming orders are handled here until they are shipped or canceled. Administrators can interact with the Fulfillment Center to influence the processing of orders, as well as suppliers can use the supplier section in the Order Fulfillment Center to fill up the stock.

The whole Petstore application consists of five web applications:

- PetStore
- Order Processing Center (OPC)
- Admin
- Supplier
- Web Application Framework (WAF)

Those components are independent Enterprise Archives (EAR), which are executable files for a J2EE server. The following chapters will focus on the PetStoreEAR. It is based on the WAF which helps to separate data and presentation with a Model-View-Controller pattern. The PetStoreEAR contains the web shop, the shopping cart and the user/customer management.

In the Order Processing Center customers' orders are checked, processed and shipped. Since this event can take a while, the coupling between the PetStore and the OPC is done by a message queue (JMS). The store sends an order to the queue and is able to return immediately afterwards displaying the user that the order will be processed. The OPC receives the order by the message service.

The AdminEAR is used for administrative tasks in the Petstore. The administrator can log in, view, refuse orders and check credit card details. The supplier is able to fill the Petstore stock.

As described in Chapter 2 a J2EE application usually follows a three-tier architecture (or more tiers) with a client, middle and EIS tier. As the client is an ordinary web browser and the application data is stored in a database (EIS tier), only the middle tier will be discussed here.

4.1.1 EJBs in Petstore

The whole Petstore application contains 31 Enterprise JavaBeans. The distribution in the different components is depicted in Table 6. The session beans are distinguished between stateless (sl) and stateful (sf). The following description concentrates on the PetstoreEAR.
The 8 entity beans in the Petstore enterprise archive are mainly customer related. They store information like credit card details, contact information and address. In detail, the entity beans are:

- **Account**
  Keeps information about the status of an account.

- **Address**
  Contains information about street, city, state, zip code and country of a customer.

- **ContactInfo**
  Email address, telephone number and names are stored here.

- **Counter**
  A number for customer orders is stored in this bean.

- **CreditCard**
  Holds the expiry date, card number and type.

- **Customer**
  Contains an user identifier for a customer.

- **Profile**
  The preferred language, a favourite category and preferences for a list and banner are held here.

- **User**
  Contains the user name and password for a Petstore user

The 7 session beans provide the application with different services:

- **AsyncSender (stateless)**
  Sends orders over an asynchronous messaging server to the Order Processing Center (OPC).

- **Catalog (stateless)**
  Makes queries to the database. Note: This bean is not used in the default configuration of Petstore. Instead, a CatalogDAO pattern is used (see chapter 2.4.4 for more information).

- **SignOn (stateless)**
  Authenticates users, who are trying to log in the shop.
• UniqueIdGenerator (stateless)
  Provides an order with an unique identifier (increases number stored in the Counter entity bean afterwards)
• ShoppingCart (stateful)
  The actual shopping cart for a certain user.
• ShoppingClientFacade (stateful)
  Provides a facade to the EJBs related to a shopping client.
• ShoppingController (stateful)
  Is the first contact in the EJB layer and processes requests from the web layer.

Figure 12 shows the interconnection between entity beans. Besides, it shows the persistent attributes of those beans.
In comparison to the entity beans the session bean’s interconnections are fairly simple. The UniqueIdGenerator is connected to the entity bean Counter which stores an application wide unique number. The stateless SignOn service accesses the User entity bean for authentication purposes. The stateful ShoppingCart needs to get product information from the stateless session bean Catalog. Figure 13 shows this as a class diagram.

4.1.2 Web to EJB

The way from a mouse click in the web interface down to a EJB method call has several stops in which message conversations or security checks are made. Figures 14 and 15 show this as an UML sequence diagram.

A certain user gesture results in a message from the client machine to the server over the web. The message is received by the server side MainServlet as the type HttpServletRequest and contains informations about possible form data, HTTP request headers or the client’s hostname. This servlet forwards the request to a RequestProcessor and selects the next visible web-page view for the user once the request treatment is finished. This object belongs to the central delivery point for all client requests (see also chapter 2.4.1 for the front controller pattern), and it analyses the client request with the help of a HTMLAction object to extract a certain action from the HTTP header. A certain HTMLAction can fulfil tasks which are necessary for the web layer and its return value can be an application Event. If the return value is Null there will be no further processing in the EJB layer. These events are designed in the Web Application Framework (WAF) by the Java
Blueprints developers. A well-known open-source web application framework is, for example, the Apache Struts project. Application events objects contain a member `ejbActionClassName` which specifies the EJB class to handle this event.

This newly created event is then passed on by the `RequestProcessor` to a `WebController` which is an additional layer (Figure 14), mainly to synchronize method calls to the following stateful session bean `EJBController` (Figure 15). Moreover, at this point the web and the EJB tier part. Finally, the `EJBController` converts events, with the help from the `StateMachine`, to commands, which will be executed by command handlers (`EJBAction` objects). The `EJBController` passes the event to the `StateMachine` and the latter is, in this way, completely independent from the events or EJB components. It ties all events dynamically together at run-time thus providing support for reusable components.
4.2 Compiling and Debugging

4.2.1 Ant

Ant is a Java-based build tool, developed as an open source tool by the Jakarta Project (http://jakarta.apache.org/). The aim is the same as the well known Unix tool make, but ant was adapted to integrate well into the Java language. The configuration is based on XML files.

Ant's configuration needed to be adapted for the logging with Log4j and AspectJ and is described in the following chapters.

4.2.2 Log4j

This open source project has been developed to put log statements in the source code of an application and it can be configured during run-time by external configuration files. With these Java property or XML configuration files it is feasible to enable different log levels. The different levels make it possible to get only those log information, which are interesting for the current purpose of the developer. This is a difference and also an improvement to the plain System.out.println call. Of course, the developer has to decide which log level an output should belong to.

One advantage of log4j is that the log statements can remain in the shipped application code without incurring a high performance cost, because Log4j is designed to keep a balance between flexibility and speed. The Log4j project provides an API, which developers can use to implement log output for the different log levels. Log4j has three main components:
1. Logger

The logger object is the main component of log4j. A logger will take a log request and handle it. On top of the logger hierarchy is a so-called root logger. All created loggers will inherit from this object, so that also a class without an individual logger could always call the root logger (not recommended).

For the creation of a logger a static method of the Logger class must be called. If a logger with the specified name already exists, it will be returned, otherwise it will be created first.

2. Appender

The appender object decides where to print the log information. Log4j supports writing to different destinations by default. Appenders exist for writing to the console, files, GUI components, remote socket servers, JMS, Windows NT Events and remote UNIX Syslog daemons. It is also possible to log to several destinations by attaching all the desired appenders to one logger.

3. Layout

The layout object is responsible for the style and content of the output log. Different information can be included, like date, time, logger information, etc. For specifying the output format, a class PatternLayout is used, and its utilization is similar to the C language printf function. A layout can be applied to by connecting it to an appender object.

Starting with SUN's J2SE SDK 1.4 a logging API is included in the distribution. But this API stays behind the possibilities of log4j and also the configuration is less comfortable. In log4j it is feasible to include the configuration information directly into the source code. However, it is more convenient to write an external configuration file to control the log output at run-time.

To enable the logging in Petstore several preconditions have to be met. A properties file includes the information where the logging should be displayed (in this case file and console). Furthermore, the log level and the log format is described. The properties file is shown in SourceCode 1.
# Set root logger level to DEBUG and its appender to A1.
log4j.rootLogger=DEBUG, A1, file
# A1 is set to be a ConsoleAppender.
log4j.appender.file=org.apache.log4j.FileAppender
# A1 uses PatternLayout.
log4j.appender.A1.layout.ConversionPattern=%-8r [%t] %-5p %-50c %x - %m%n
log4j.appender.file.File=D:\petstorelog.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%-8r [%t] %-5p %-5c %x - %m%n

SourceCode 1 Log4j properties file

In file MainServlet.java the .properties file is loaded in the init() method of the Servlet. When this method is executed the logging starts. SourceCode 2 shows the loading command.

```java
PropertyConfigurator.configure("D:\\project\\petstore1.3.1\\src\\log4j.properties");
```

SourceCode 2 load Log4j properties file in MainServlet.java

The log4j jar file (log4j-1.2.7.jar) was copied into the Petstore source code folder, for example, into %petstore_home%\src\lib\log4j. SourceCode 3 shows an excerpt from an example build.xml file. There, it can be seen how to add the log4j specific information for the ant build process.

```xml
<property name="log4j.classpath" value="../../../lib/log4j/log4j-1.2.7.jar"/>
<property name="address.classpath" value="${address.classbindir}:${log4j.classpath}:${j2ee.classpath}"

SourceCode 3 include Log4j in ant's build.xml files

4.2.3 AspectJ

Aspect-Oriented Programming (AOP) enhances other programming models, mainly Object-Oriented Programming (OOP) but also procedural programming. AOP adds new functionality to those programming paradigms to solve cross-cutting problems. Cross-cutting problems occur when problem domains, like logging, security or error handling, have to be considered. Such problem domains destroy the clean modularity in an application because logging code for example has to be inserted into modules from other problem domains. The result is “tangled” code that is difficult to maintain.
Therefore, AOP distinguishes between the new *aspects* and the components (classes) which are known from OOP. System units which can be cleanly encapsulated are referred as components. Wherever this clean encapsulation fails, aspects can help to avoid cross-cut code. MIT Technology Review lists AOP as one of “ten emerging areas of technology that will soon have a profound impact on the economy and on how we live and work”.

AspectJ is an implementation of AOP for Java. It adds a few new constructs to the Java programming language. The most important are: aspects, join points, pointcuts, and advice. An *aspect* is AspectJ’s unit of modularity, like the keyword *class* in Java. Join points are “points in the execution” of Java programs. Such points exists for method calls, method executions or for catching exceptions and so on. Pointcuts are collections of join points that have “something” in common. For example:

```java
pointcut publicMethodCall(): call( * com.sun.j2ee.blueprints..*,*(..))
```

When a described pointcut matches a join point, it gets recognized and a user defined action takes place. Here advices become involved. Advices can be imagined as the methods of AspectJ. An advice is attached to a pointcut and is executed when a particular pointcut applies. The keywords *before*, *after* and *around* decide when the advice is performed. *Before* and *after* are straightforward; they execute before or after a join point is traversed. *Around* intercepts the original method call and is called instead. The developer can write code which is executed before the method call. Then, he has to call the actual method with the keyword `proceed()` and the remaining advice code is performed after the method call.

AspectJ provides tools for the automated build process with ant. To make AspectJ available in Petstore, a few changes in some configuration files are needed. SourceCode 4 shows how to make the AspectJ compiler `ajc` available.

```xml
<taskdef name="ajc"
    classname="org.aspectj.tools.ant.taskdefs.Ajc" >
    <!-- declare classes needed to run the tasks and tools -->
    <classpath>
        <pathelement location="lib/ant/aspectj/aspectj-ant.jar"/>
        <pathelement location="lib/ant/aspectj/aspectjtools.jar"/>
        <pathelement location="lib/ant/jdk13/tools.jar"/>
    </classpath>
</taskdef>
```

*SourceCode 4 AspectJ addition to file: %petstore_home%\src\build.xml*
Once ajc is known, in every build.xml file the standard Java compiler needs to be replaced by ajc. SourceCode 5 depicts a part of an example build.xml file in which ajc is used for compilation. Remarkably, nothing else needs to be changed to get AspectJ working.

```xml
<target name="compile" depends="init">
    <mkdir dir="${petstore.classbindir}"/>
    <!-- Petstore Classes -->
    <ajc srcdir="${petstore.src}"
        destdir="${petstore.classbindir}"
        classpath="${petstore.classpath}"
        includes="com/**">
        <classpath>
            <pathelement location="${aspectj.classpath}"/>
        </classpath>
    </ajc>
</target>
```

*SourceCode 5 Every build.xml needs the AspectJ compiler ajc*

Some problems occurred before AspectJ started working. Ant had to be updated to version 1.5 (or above) to avoid errors during compile time. A new Ant jar file in the directory %petstore_home%/src/lib/ant\ solved this problem.

Another problem happened during deployment time. Although every build.xml file was changed, the AspectJ run time jar file could not be found by some classes. The only solution was to include the jar file into the applications executable Enterprise ARchive file (EAR).

### 4.2.4 IDE

To enable the IDE to find the Java source files during the debugging process, the modularized file structure was changed to the common Java package/file structure, like /com/sun/j2ee/... (or com.sun.j2ee... as package). For the sake of simplicity the original Ant build process to compile the whole application was kept. That means that the source files existed twice, but maybe as two different versions. To stay up to date, a batch script cppssrc (CoPy PetStore Source) was written which copies the source files from the Ant structure to the package structure. It expects that two environment variables are set:

- `%petstore_home%` - the Petstore home directory
- `%debug_src_files%` - the home directory for the debug source files

These variables can be overwritten by command line arguments with the following syntax:

```
cppssrc petstore_home_dir debug_src_files
```

Example:
4.3 Business Logic

When customers access the Petstore there are different scenarios. The following sub chapters present those having an impact on the business logic of the application. As the general way from a user click to the business logic is always the same, as described in chapter 4.1.2, the sequence diagrams usually start with the EJBACTIONS. They are the actual objects which access the business logic.

4.3.1 Browsing the shop

When customers are browsing through the Petstore shop, there is no enterprise bean involved. JavaServer Pages directly access an object called CatalogHelper which in turn gets data from the database through a Data Access Object CatalogDAO (see chapter 2.4.4 for the DAO pattern).

The design decision, why the catalog data is accessed through a DAO, has been made because of the necessity for read-only access to the shop's inventory [SUN02d]: “In such situations, using entity beans incurs a performance penalty while providing little or no additional value.”

4.3.2 Catalog

The catalog stored in a database, can be accessed in two different ways. First, as shown in Figure 16 and mentioned in chapter 4.3.1, a class CatalogHelper gets products from the database with the help of an object CatalogDAO. The Factory pattern, shown as class CatalogDAOFactory, makes it easy to exchange the data sources in which the actual catalog data is stored. Therefore, the CatalogHelper requests a Data Access Object (1). The factory has knowledge of the necessary CatalogDAO and creates it (1.1). Now, object CatalogHelper sends a message to the DAO to get the desired item (2). Finally, CatalogDAO gets the information from the EIS tier (2.1).

The second way uses an additional session bean CatalogEJB between CatalogHelper and CatalogDAOFactory. This session bean is used by the ShoppingCartEJB for receiving items from the database. This approach is mentioned in [SUN02d] and describes the benefits of a session bean in this situation.
To decide which method should be used, the CatalogHelper gets a boolean argument useFastLane passed. This refers to a design pattern Fast Lane Reader which is used for improving performance by avoiding the employment of enterprise beans. In Petstore this pattern is implemented for illustration purposes only: “In an actual production application, the architecture would either use a fast lane pattern or it would not, instead of providing a runtime choice.” [SUN02e]

### 4.3.3 Add item to cart

When a customer shops and decides to put an item to the cart, a CartEvent is created. Figure 17 shows the sequence diagram for the first item put into the shopping cart.
The StateMachine calls the appropriate Action, CartEJBAction (1). These objects request a shopping cart from the ShoppingClientFacadeLocal (1.1). Details for the SessionFacade pattern can be found in chapter 2.4.3. The Facade creates (1.1.1) an object ShoppingCartLocal (it is the first item to be added) which is then returned to the action handler. The CartEvent requests the ActionType (1.2) and the ItemId (1.3). The type could be “add item” or “update item”, depending on the user action. In this example the type was “add item” and results in a shopping cart call for adding the item to the cart (1.4).

### 4.3.4 Sign on

For certain actions a user has to be signed in. That means, the customer needs to be authenticated by the shop for updating personal information and preferences or if an order is made. The sign on process is partly described in chapter 2.4.2 because of the here used design pattern Intercepted Filter. If the SignOnFilter makes an authentication request (1), depicted in Figure 18, the stateless session bean SignOn will look for the user name by making a findByPrimaryKey() call to the
EJBContainer (1.1). If the user cannot be found, a FinderException will return the boolean value false. Otherwise, the password entered by the customer is compared with the one stored in the User entity bean (1.2).

4.3.5 Submit orders

After shopping the customer wants to make an order and checks out. The customer’s billing and shipping information are presented for possible changes in those settings. A click on the submit button makes the order binding.

The business action OrderEJBAction, shown in Figure 19, gets the billing and shipping information from the OrderEvent (1-3). In the next step, a stateless session bean UniqueIdGenerator is accessed (4) to get a unique identifier for the purchase order made. Therefore, the bean requests a counter from an entity bean (4.1). Every access to the Counter bean increments the stored value. After receiving the user ID from the stateful session bean ShoppingClientFacade (5), an object PurchaseOrder is created (6) and the previous collected variables are set (7). When the ShoppingClientFacade returned the ShoppingCart reference (8), OrderEJBAction adds all ordered items from the shopping cart (9) to the PurchaseOrder (10). In the last step the PurchaseOrder is serialized to an XML string and this text is send to the stateless session bean AsyncSender. This object only provides one service method sendAMessage() to pass the order to a JMS queue. Therefore, AsyncSender connects to the suitable queue and creates a sender object for it. Then the message is send and JMS delivers it.

Figure 18 sequence diagram: sign on
Figure 19: Sequence diagram: submit orders
4.3.6 Update account information

Only a signed on customer is able to edit its account information. After changing the user has to press a submit button. In Figure 20, the StateMachine calls the matching action, here CustomerEJBAction (1), which in turn requests the ShoppingClientFacade for this customer. Then, the CustomerEJBAction gets the type of action (1.2). That type depends on the user's request. In case the user just registered the ActionType would be "create". But here, the user only changed its personal information, therefore, the ActionType is "update". From the ShoppingClientFacade the action handler gets the Customer reference delivered after user identifier was looked up by the findByPrimaryKey() method (1.3.1). With access to the user's entity bean object CustomerEJBAction can save the personal information (1.4 – 1.6).

Figure 20 sequence diagram: update account information
4.3.7 Change Locale

The Locale object contains the language and country information of the web shop. A user can switch from one language to another. To guarantee also that information from the business logic are displayed in the chosen language, a ChangeLocaleEJBAction takes care of the update (Figure 21).

The customer's shopping cart is the only object which needs to be notified of the Locale change. First, the action handler gets the Locale from ChangeLocaleEvent (1.1). After storing the regional information in theStateMachine (1.2) and receiving the ShoppingCart from it (1.3) the ChangeLocaleEJBAction informs the cart about the change (1.5).

*Figure 21 sequence diagram: change locale*
5 Summary

Originally, performance had not been considered during the development of Petstore and was thus highly criticized by the Java community. Yet, especially after the Microsoft Corporation took the original Petstore application as a foundation stone for performance measurements of their own .NET Petshop, performance optimized versions of Petstore have been developed (JPetstore [BEG02]). It could be interesting to have a look at the different versions as well and see how the optimizations are done. These information could probably flow back into the COMPAS framework to make suggestions for design improvements.

The scenarios which access the Enterprise JavaBeans have been identified and necessary sequence diagrams for a comparison were created. The effort to concentrate on the business logic and especially EJBs keeps the diagrams fairly simple. Since during the work on this thesis the modeling part of COMPAS is not yet implemented, this approach has to be reconsidered to be sufficient.

For the benchmark test with Trade 3 further measurements are necessary. The fundamental adjustments are made but only for the synchronous order processing mode. Therefore no message driven beans have been involved in the benchmark. Yet, as COMPAS is able to monitor that kind of EJBs as well, a benchmark with MDB has to be done. The workload scripts also make use of five of ten actions only (login, logout, buy, sell, portfolio). Workload which uses all ten actions could lead to different results because of changed EJB access. Furthermore, the only boundary for more workload and more virtual users respectively was the harddisk bottleneck. The RAM equipment did not seem to be the cause as monitoring showed. It would also be helpful to know which process is responsible for the high write activity. If the problem can be enclosed like this, a little tuning at those hardware parts could result in a more regular utilization of the whole server. Once that these issues have been cleared, a benchmark run over a longer time period should be started to have reliable measurements for comparison.
# 6 Appendix

## 6.1 Logging aspect

The following source code shows the used aspect to log all the Petstore specific method calls. All those methods are in the package `com.sun.j2ee.blueprints`.

```java
package ie.dcu.pel.globalaspects;
import org.apache.log4j.Logger;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.reflect.CodeSignature;

aspect LoggerAspect {
    static Logger logger = Logger.getLogger(" ");
    pointcut publicMethodCall(): call( * com.sun.j2ee.blueprints..*.*(..)) && !within(LoggerAspect);

    /** This is the advice code on the defined set of pointcuts in
     * publicMethodCall() - definition.
     * Wrap the method call so we can log that it is about to be run,
     * the parameters it will run with and the result we get from it.
     * "around" is instead of the method call, alternatives are
     * "before" or "after".
     */
    Object around(): publicMethodCall() {
        // Log information about the currently intercepted join point
        // thisJoinPoint is a special construct in AspectJ offering
        // reflective properties for the join point.
        logger.info("Log alert for " + thisJoinPoint);
        // Print the method parameters by calling the method below
        printParameters(thisJoinPoint);
        // Execute the wrapped method and catch the result
        Object result = proceed();
        // Print result, if it is not null
        if (result != null)
            logger.info("Result of " + thisJoinPoint + ": " + result.toString());
        return result;
    }

    /** Print the method input parameters
     * @param jp the join point a pointcut has caught an event on
     */
    static private void printParameters(JoinPoint jp) {
```
StringBuffer buf = new StringBuffer();
    // Reflective access to join point parameters below
    Object[] args = jp.getArgs();
    String[] names = ((CodeSignature)jp.getSignature()).getParameterNames();
    Class[] types = ((CodeSignature)jp.getSignature()).getParameterTypes();
    buf.append("Arguments: \n");
    // If there are any arguments, print them
    if (args.length > 0) {
      // Go through the arguments and add them to the StringBuffer
      for (int i = 0; i < args.length; i++) {
        buf.append(" " + i + "." + names[i] + ": " +
          types[i].getName() + " = " + args[i] + 
            \n");
      }
      // Log the contents of buf
      logger.info(buf.toString());
    }

6.2 Batchfile Copy Petstore files

This shows the source code of the windows batchfile which copies Petstore's source files into a package based structure.

@echo on
if "%OS%" == "Windows_NT" setlocal
rem -----------------------------------------------
rem Script cppssrc to copy java source files from the ant module structure
rem to a package structure the SunONE Studio understands
rem
rem Environment Variable Prequisites
rem
rem PETSTORE_HOME Must point at your Petstore Home Directory.
rem DEBUG_SRC_FILES Must point at your directory for the packet structure
rem
rem $Thomas Wermter V0.07 $
rem -----------------------------------------------
rem Test if Command line arguments are set, else look for environment variables
if not "%1"=="" if not "%2"=="" goto gotDirs
rem Make sure prerequisite environment variables are set
if not "%PETSTORE_HOME%" == "" if not "%DEBUG_SRC_FILES%" == "" goto gotEnv
echo The PETSTORE_HOME AND/OR DEBUG_SRC_FILES environment variable is not defined
echo nor did you set commandline parameters like:
echo %0 c:\petstore1.3.1 c:\debugfiles
goto end
rem Set Variables
:gotEnv
SET FROM=%PETSTORE_HOME%
SET TO=%DEBUG_SRC_FILES%
goto StartCopy
:gotDirs
SET FROM=%1
SET TO=%2

:StartCopy
xcopy /SEY "%FROM%\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\apps\admin\src\admin\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\apps\admin\src\client\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\apps\opc\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\apps\petstore\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\apps\supplier\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\address\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\asyncsender\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\cart\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\catalog\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\contactinfo\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\creditcard\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\customer\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\encodingfilter\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\lineitem\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\mailer\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\processmanager\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\purchaseorder\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\servicelocator\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\signon\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\supplierpro\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\uidgen\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\util\tracer\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\components\xmldocuments\src\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\waf\src\controller\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\waf\src\view\template\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\waf\src\view\taglibs\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\webservices\apps\admin\src\client\com" "%TO%\com\"
xcopy /SEY "%FROM%\src\webservices\apps\admin\src\com" "%TO%\com\"
6.3 OpenSTA workload scripts

This chapter includes the 3 OpenSTA workload generation scripts. As they itself are already long enough, the included usernumber enumeration has been shortened. The numbers are counted up in steps of three and the OpenSTA tool is able to generate them automatically.

6.3.1 Buy stock script

!Browser: NETSCAPE 4.7
!Date: 10/02/2003

Environment

Description ""
Mode HTTP
Wait UNIT MILLISECONDS

Definitions

! Standard Defines
Include "RESPONSE_CODES.INC"
Include "GLOBAL_VARIABLES.INC"
CHARACTER*512 USER_AGENT
CHARACTER*256 MESSAGE
CHARACTER*512 MY_USERNUMBER, LOCAL
Integer USE_PAGE_TIMERS
Timer T_T3BUY
CHARACTER*1024 cookie_3_0
CHARACTER*1024 cookie_12_0
CHARACTER*512 USERNUMBER_BUY ( "1", "4", "7" &
, "10", "13", "16" &
![]):...
CONSTANT DEFAULT_HEADERS = "Host: websphere4x:9080^J" &
"Accept-Encoding: gzip^J" &
"User-Agent: Mozilla/4.78 [en] (Windows NT 5.0; U)"

Code

!Read in the default browser user agent field
Entry[USER_AGENT,USE_PAGE_TIMERS]
Start Timer T_T3BUY
PRIMARY GET URI "http://websphere4x:9080/trade/app HTTP/1.0" ON 1 &
HEADER DEFAULT_HEADERS &
,WITH {"Connection: Keep-Alive", &
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &

"Accept-Language: en",
"Accept-Charset: iso-8859-1,*utf-8")

DISCONNECT FROM 1

!Autogenerated (but rounded) think time
WAIT 300

!Reduced stress load think time
!WAIT 100

PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 2 &
HEADER DEFAULT_HEADERS &
,WITH {"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*utf-8")

DISCONNECT FROM 2

WAIT 10000

!WAIT 100

ACQUIRE TEST-WIDE MUTEX "LOGIN"

NEXT USERNUMBER_BUY

SET MY_USERNUMBER = USERNUMBER_BUY

RELEASE TEST-WIDE MUTEX "LOGIN"

LOG MY_USERNUMBER ," login (BUY)"

!LOGIN

PRIMARY POST URI "http://websphere4x:9080/trade/app HTTP/1.0" ON 3 &
HEADER DEFAULT_HEADERS &
,WITH{"Referer: http://websphere4x:9080/trade/app", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*utf-8", &
"Content-type: application/x-www-form-urlencoded", &
"Content-length: 35"}

,BODY "uid=uid%3A"+MY_USERNUMBER+"&passwd=xxx&action=login"

Load Response_Info Header on 3 &
Into cookie_3_0 &
,WITH "Set-Cookie,JSESSIONID"

DISCONNECT FROM 3

WAIT 10000

!WAIT 1000

PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 4 &
HEADER DEFAULT_HEADERS &
,WITH{"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*utf-8", &
"Cookie: "+cookie_3_0}
DISCONNECT FROM 4
WAIT 10000
!WAIT 1000
PRIMARY GET URI "http://websphere4x:9080/trade/images/arrowdown.gif HTTP/1.0" ON 5 &
   HEADER DEFAULT_HEADERS &
   ,WITH {"Referer: http://websphere4x:9080/trade/app", &
   "Connection: Keep-Alive", &
   "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png", &
   "Accept-Language: en", &
   "Accept-Charset: iso-8859-1","utf-8", &
   "Cookie: "+cookie_3_0}
DISCONNECT FROM 5
WAIT 300
!WAIT 100
PRIMARY GET URI "http://websphere4x:9080/trade/images/arrowup.gif HTTP/1.0" ON 6 &
   HEADER DEFAULT_HEADERS &
   ,WITH {"Referer: http://websphere4x:9080/trade/app", &
   "Connection: Keep-Alive", &
   "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png", &
   "Accept-Language: en", &
   "Accept-Charset: iso-8859-1","utf-8", &
   "Cookie: "+cookie_3_0}
DISCONNECT FROM 6
WAIT 12000
!WAIT 2000
GENERATE STOCKNUMBER
LOG "Stock-BUY: ", STOCKNUMBER
!Quote
PRIMARY GET URI &
   "http://websphere4x:9080/trade/app?action=quotes&symbols=s%3A"+STOCKNUMBER+" &
   HTTP/1.0" ON 7 &
   HEADER DEFAULT_HEADERS &
   ,WITH {"Referer: http://websphere4x:9080/trade/app", &
   "Connection: Keep-Alive", &
   "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
   "Accept-Language: en", &
   "Accept-Charset: iso-8859-1","utf-8", &
   "Cookie: "+cookie_3_0}
DISCONNECT FROM 7
WAIT 1000
!WAIT 100
PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 8 &
   HEADER DEFAULT_HEADERS &
WITH ("Connection: Keep-Alive",
    &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,* utf-8", &
"Cookie: "+cookie_3_0)

DISCONNECT FROM 8
WAIT 10000
!WAIT 1000
!BUY

PRIMARY GET URI &
"http://websphere4x:9080/trade/app?action=buy&symbol=s%3A"+STOCKNUMBER+ &
"&quantity=100 HTTP/1.0" ON 9 &
HEADER DEFAULT_HEADERS &
,WITH ("Referer: &
http://websphere4x:9080/trade/app?action=quotes&symbols=s%3A"+STOCKNUMBER,&
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*",&
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,* utf-8", &
"Cookie: "+cookie_3_0)

DISCONNECT FROM 9
WAIT 11000
!WAIT 2000

PRIMARY GET URI "http://websphere4x:9080/trade/app?action=portfolio HTTP/1.0" &
ON 10&
HEADER DEFAULT_HEADERS &
,WITH ("Referer: &
http://websphere4x:9080/trade/app?action=buy&symbol=s%3A"+STOCKNUMBER &
"&quantity=100", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*",&
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,* utf-8", &
"Cookie: "+cookie_3_0)

DISCONNECT FROM 10
WAIT 700
!WAIT 200

PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 11 &
HEADER DEFAULT_HEADERS &
,WITH ("Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*",&
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,* utf-8", &
"Cookie: "+cookie_3_0)
DISCONNECT FROM 11
WAIT 11000
!WAIT 1000
!LOGOUT
PRIMARY GET URI "http://websphere4x:9080/trade/app?action=logout HTTP/1.0" ON 12 & HEADER DEFAULT_HEADERS &
,WITH {"Referer: http://websphere4x:9080/trade/app?action=portfolio", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,utf-8", &
"Cookie: Cookie_3_0}"
Load Response_Info Header on 12 &
Into cookie_12_0 &
,WITH "Set-Cookie,JSESSIONID"
DISCONNECT FROM 12
WAIT 700
!WAIT 100
PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 13 & HEADER DEFAULT_HEADERS &
,WITH {"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,utf-8", &
"Cookie: Cookie_12_0}"
DISCONNECT FROM 13
LOG MY_USERNUMBER," logout (BUY)"
SYNCHRONIZE REQUESTS
End Timer T_T3BUY
Exit
ERR_LABEL:
If (MESSAGE <> "") Then
  Report MESSAGE
Endif
Exit

6.3.2 Quote stock script

!Browser:NETSCAPE 4.7
!Date: 10/02/2003
Environment

Description ""
Mode HTTP
Wait UNIT MILISECONDS
Definitions
! Standard Defines
Include "RESPONSE_CODES.INC"
Include "GLOBAL_VARIABLES.INC"

CHARACTER*512 USER_AGENT
CHARACTER*256 MESSAGE
Integer USE_PAGE_TIMERS
Timer T_T3QUOTE
CHARACTER*1024 cookie_3_0
CHARACTER*1024 cookie_13_0
CHARACTER*512 MY_USERNUMBER, LOCAL
CHARACTER*512 STOCKNUMBER_OLD, LOCAL
CHARACTER*512 USERNUMBER_QUOTE ( "0", "3", "6" &
, "9", "12", "15" &
[...]
CONSTANT DEFAULT_HEADERS = "Host: websphere4x:9080^J" &
"Accept-Encoding: gzip^J" &
"User-Agent: Mozilla/4.78 [en] (Windows NT 5.0; U)"

Code
!Read in the default browser user agent field
Entry[USER_AGENT,USE_PAGE_TIMERS]
Start Timer T_T3QUOTE
PRIMARY GET URI "http://websphere4x:9080/trade/app HTTP/1.0" ON 1 &
 HEADER DEFAULT_HEADERS$
,WITH{"Connection: Keep-Alive"," &
 "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*"," &
 "Accept-Language: en"," &
 "Accept-Charset: iso-8859-1,* utf-8")
DISCONNECT FROM 1
!Autogenerated (but rounded) think time
WAIT 300
!Reduced stress load think time
!WAIT 100
PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 2&
 HEADER DEFAULT_HEADERS$
,WITH{"Connection: Keep-Alive"," &
 "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*"," &
 "Accept-Language: en"," &
 "Accept-Charset: iso-8859-1,* utf-8")
DISCONNECT FROM 2
WAIT 10000
!WAIT 1000
ACQUIRE TEST-WIDE MUTEX "LOGIN"
NEXT USERNUMBER_QUOTE
SET MY_USERNUMBER = USERNUMBER_QUOTE
RELEASE TEST-WIDE MUTEX "LOGIN"
LOG MY_USERNUMBER ," login (Quote)"
!LOGIN

PRIMARY POST URI "http://websphere4x:9080/trade/app HTTP/1.0" ON 3&
HEADER DEFAULT_HEADERS &
,WITH {"Referer: http://websphere4x:9080/trade/app", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8", &
"Content-type: application/x-www-form-urlencoded", &
"Content-length: 35"}&
,BODY "uid=uid%3A"+MY_USERNUMBER+"&passwd=xxx&action=login"

Load Response_Info Header on 3&
Into cookie_3_0 &
,WITH "Set-Cookie,JSESSIONID"
DISCONNECT FROM 3
WAIT 20000
!WAIT 2000

PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 4&
HEADER DEFAULT_HEADERS&
,WITH {"Referer: http://websphere4x:9080/trade/app", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8", &
"Cookie: "+cookie_3_0}"
DISCONNECT FROM 4
WAIT 6000
!WAIT 300

PRIMARY GET URI "http://websphere4x:9080/trade/images/arrowdown.gif HTTP/1.0" ON 5&
HEADER DEFAULT_HEADERS &
,WITH {"Referer: http://websphere4x:9080/trade/app", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8", &
"Cookie: "+cookie_3_0}"
DISCONNECT FROM 5
WAIT 300
!WAIT 100

PRIMARY GET URI "http://websphere4x:9080/trade/images/arrowup.gif HTTP/1.0" ON 6 &

HEADER DEFAULT_HEADERS &
,WITH {"Referer: http://websphere4x:9080/trade/app", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8", &
"Cookie: "+cookie_3_0}

DISCONNECT FROM 6

WAIT 3000

!WAIT 100

GENERATE STOCKNUMBER

LOG "Stock-QUOTE: ", STOCKNUMBER

QUOTE

PRIMARY GET URI &

"http://websphere4x:9080/trade/app?action=quotes&symbols=s:"+STOCKNUMBER+ &
" HTTP/1.0" ON 7 &

HEADER DEFAULT_HEADERS &
,WITH {"Referer: http://websphere4x:9080/trade/app", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8", &
"Cookie: "+cookie_3_0}

DISCONNECT FROM 7

WAIT 400

!WAIT 100

PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 8 &

HEADER DEFAULT_HEADERS &
,WITH {"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8", &
"Cookie: "+cookie_3_0}

DISCONNECT FROM 8

SET STOCKNUMBER_OLD=STOCKNUMBER

GENERATE STOCKNUMBER

LOG "Stock-QUOTE: ", STOCKNUMBER

WAIT 15000

!WAIT 2000

QUOTE
PRIMARY GET URI &
"http://websphere4x:9080/trade/app?action=quotes&symbols=s:"+STOCKNUMBER+ &
" HTTP/1.0" ON 9&
HEADER DEFAULT_HEADERS &
,WITH {"Referer: &
http://websphere4x:9080/trade/app?action=quotes&symbols=s%3A"+STOCKNUMBER_OLD,&
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*",utf-8", &
"Cookie: "+cookie_3_0}
DISCONNECT FROM 9
WAIT 400
!WAIT 100
PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 10 &
HEADER DEFAULT_HEADERS &
,WITH {"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*",utf-8", &
"Cookie: "+cookie_3_0}
DISCONNECT FROM 10
SET STOCKNUMBER_OLD=STOCKNUMBER
GENERATE STOCKNUMBER
LOG "Stock-QUOTE: ", STOCKNUMBER
WAIT 10000
!WAIT 1000
!QUOTE
PRIMARY GET URI &
"http://websphere4x:9080/trade/app?action=quotes&symbols=s:"+STOCKNUMBER+ &
" HTTP/1.0" ON 11&
HEADER DEFAULT_HEADERS &
,WITH {"Referer: &
http://websphere4x:9080/trade/app?action=quotes&symbols=s"+STOCKNUMBER_OLD,&
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*",utf-8", &
"Cookie: "+cookie_3_0}
DISCONNECT FROM 11
WAIT 300
!WAIT 100
PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 12 &
 HEADER DEFAULT_HEADERS &
 ,WITH {"Connection: Keep-Alive","Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*","Accept-Language: en","Accept-Charset: iso-8859-1,*,utf-8","Cookie: "+cookie_3_0}
 DISCONNECT FROM 12
 WAIT 5000
 !WAIT 500
 !LOGOUT
 PRIMARY GET URI "http://websphere4x:9080/trade/app?action=logout HTTP/1.0" ON 13 &
 HEADER DEFAULT_HEADERS &
 Load Response_Info Header on 13 &
 Into cookie_13_0 &
 ,WITH "Set-Cookie,JSESSIONID"
 DISCONNECT FROM 13
 WAIT 300
 !WAIT 100
 PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 14 &
 HEADER DEFAULT_HEADERS &
 ,WITH {"Connection: Keep-Alive","Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*","Accept-Language: en","Accept-Charset: iso-8859-1,*,utf-8","Cookie: "+cookie_13_0}
 DISCONNECT FROM 14
 LOG MY_USERNUMBER," logout (Quote)"
 SYNCHRONIZE REQUESTS
 End Timer T_T3QUOTE
 Exit
 ERR_LABEL:
   If (MESSAGE <> "") Then
     Report MESSAGE
   Endif
Exit

6.3.3 Sell stock script

!Browser: NETSCAPE 4.7
!Date: 10/02/2003

Environment
  Description ""
  Mode HTTP
  Wait UNIT MILLISECONDS

Definitions
  ! Standard Defines
  Include "RESPONSE_CODES.INC"
  Include "GLOBAL_VARIABLES.INC"
  CHARACTER*512 USER_AGENT
  CHARACTER*256 MESSAGE
  CHARACTER*512 MY_USERNUMBER, LOCAL
  Integer USE_PAGE_TIMERS
  Timer T_T3SELL
  CHARACTER*1024 cookie_3_0
  CHARACTER*1024 cookie_12_0
  CHARACTER*512 NO_OF_HOLDINGS, LOCAL
  CHARACTER*512 HOLDING_ID, LOCAL
  CHARACTER*512 USERNUMBER_SELL ("2", "5", "8" &
  , "11", "14", "17" &
  [...]
  CONSTANT DEFAULT_HEADERS = "Host: websphere4x:9080^J"
  "Accept-Encoding: gzip^J" &
  "User-Agent: Mozilla/4.78 [en] (Windows NT 5.0; U)"

Code

!Read in the default browser user agent field
Entry[USER_AGENT, USE_PAGE_TIMERS]
Start Timer T_T3SELL
PRIMARY GET URI "http://websphere4x:9080/trade/app HTTP/1.0" ON 1 &
  HEADER DEFAULT_HEADERS &
  ,WITH{"Connection: Keep-Alive", &
  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, &
  image/png, */*", &
  "Accept-Language: en", &
  "Accept-Charset: iso-8859-1,*,utf-8"}
DISCONNECT FROM 1
WAIT 400
!WAIT 100

PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 2&
  HEADER DEFAULT_HEADERS&
  ,WITH ("Connection: Keep-Alive", &
  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, &
   image/png, */*",&
  "Accept-Language: en", &
  "Accept-Charset: iso-8859-1,*,utf-8")
DISCONNECT FROM 2
WAIT 10000
!WAIT 1000

ACQUIRE TEST-WIDE MUTEX "LOGIN"
  NEXT USERNUMBER_SELL
  SET MY_USERNUMBER = USERNUMBER_SELL
RELEASE TEST-WIDE MUTEX "LOGIN"
LOG MY_USERNUMBER ," login (SELL)"
!LOGIN

PRIMARY POST URI "http://websphere4x:9080/trade/app HTTP/1.0" ON 3 &
  HEADER DEFAULT_HEADERS&
  ,WITH ("Referer: http://websphere4x:9080/trade/app", &
  "Connection: Keep-Alive", &
  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
   image/png, */*",&
  "Accept-Language: en", &
  "Accept-Charset: iso-8859-1,*,utf-8",&
  "Content-type: application/x-www-form-urlencoded", &
  "Content-length: 35")&
  ,BODY "uid=uid%3A"+MY_USERNUMBER+"&passw=xxx&action=login"
Load Response_Info Header on 3 &
  Into cookie_3_0 &
  ,WITH "Set-Cookie,JSESSIONID"
DISCONNECT FROM 3
WAIT 10000
!WAIT 1000

PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 4&
  HEADER DEFAULT_HEADERS &
  ,WITH ("Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8", &
"Cookie: "+cookie_3_0} 
DISCONNECT FROM 4 
WAIT 10000 
!WAIT 1000 
DISCONNECT FROM 5 
WAIT 400 
!WAIT 100 
DISCONNECT FROM 6 
WAIT 2000 
!WAIT 200 
!VIEW PORTFOLIO 
PRIMARY GET URI "http://websphere4x:9080/trade/app?action=portfolio & HTTP/1.0" ON 7 & HEADER DEFAULT_HEADERS & ,WITH {"Referer: http://websphere4x:9080/trade/app", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, &
image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8", &
"Cookie: "+cookie_3_0}
LOAD RESPONSE_INFO BODY ON 7 &
INTO NO_OF_HOLDINGS &
,WITH "HTML(0)/BODY(1)/TABLE(1)/TBODY(0)/TR(0)/TD(0)/TABLE(0)/TBODY(0) &
/TR(0)/TD(1):TEXT:(1)"
!If no shares available, there's nothing to sell - therefore logout
IF ( NO_OF_HOLDINGS="0" ) GOTO logout
!Get number for selling share
LOAD RESPONSE_INFO BODY ON 7 &
INTO HOLDING_ID &
,WITH "HTML(0)/BODY(1)/TABLE(1)/TBODY(0)/TR(0)/TD(0)/TABLE(0)/TBODY(0)&
/TR(1)/TD(0)/TABLE(1)/TBODY" &
"(1)/TR(1)/TD(0):TEXT:(0)"
DISCONNECT FROM 7
LOG "Stock-SELL: ", HOLDING_ID
WAIT 400
!WAIT 300
PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 8&
HEADER DEFAULT_HEADERS &
,WITH ("Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, &
image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8", &
"Cookie: "+cookie_3_0)
DISCONNECT FROM 8
WAIT 10000
!WAIT 1000
!SELL share
PRIMARY GET URI &
"http://websphere4x:9080/trade/app?action=sell&holdingID="+HOLDING_ID+&
" HTTP/1.0" ON 9&
HEADER DEFAULT_HEADERS &
,WITH ("Referer: http://websphere4x:9080/trade/app?action=portfolio", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpg, image/png, &
image/png, */*,",&
"Accept-Language: en",&
"Accept-Charset: iso-8859-1,*,utf-8",&
"Cookie: "+cookie_3_0}
DISCONNECT FROM 9
WAIT 10000
!WAIT 1000
PRIMARY GET URI "http://websphere4x:9080/trade/app?action=portfolio & HTTP/1.0" ON 10 &
HEADER DEFAULT_HEADERS &
,WITH ("Referer: &
http://websphere4x:9080/trade/app?action=sell&holdingID="+ &
HOLDING_ID,&
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpg, image/png, &
image/png, */*,",&
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8",&
"Cookie: "+cookie_3_0}
DISCONNECT FROM 10
WAIT 400
!WAIT 100
PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 11&
HEADER DEFAULT_HEADERS &
,WITH ("Connection: Keep-Alive",&
"Accept: image/gif, image/x-xbitmap, image/jpg, image/png, &
image/png, */*,",&
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,*,utf-8",&
"Cookie: "+cookie_3_0}
DISCONNECT FROM 11
WAIT 3000
!WAIT 300

logout:
PRIMARY GET URI "http://websphere4x:9080/trade/app?action=logout
HTTP/1.0" &ON 12&
HEADER DEFAULT_HEADERS &
,WITH ("Referer: http://websphere4x:9080/trade/app?action=portfolio", &
"Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,* utf-8", &
"Cookie: "+cookie_3_0)
Load Response_Info Header on 12&
Into cookie_12_0 &
,WITH "Set-Cookie, JSESSIONID"
DISCONNECT FROM 12
WAIT 400
!WAIT 100
PRIMARY GET URI "http://websphere4x:9080/trade/style.css HTTP/1.0" ON 13&
HEADER DEFAULT_HEADERS &
,WITH ("Connection: Keep-Alive", &
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, &
image/png, */*", &
"Accept-Language: en", &
"Accept-Charset: iso-8859-1,* utf-8", &
"Cookie: "+cookie_12_0)
DISCONNECT FROM 13
LOG MY_USERNUMBER"," logout (SELL)"
SYNCHRONIZE REQUESTS
End Timer T_T3SELL
Exit
ERR_LABEL:
If (MESSAGE <> "") Then
  Report MESSAGE
Endif
Exit
Bibliography


[SUN02d] Brydon, Sean: Architecture of the Sample Application. 2002,
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/sample-app/sample-app1.3.1a3.html

[SUN02b] Sun Microsystems: J2EE Patterns Catalog. 2002,
http://java.sun.com/blueprints/patterns/catalog.html


[GAM95] Gamma, Erich; Helm, Richard; Johnson, Ralf; Vlissides, John: Design Patterns: Elements of Reusable Object-Orien. , Addison-Wesley, 1995

[ALU01] Alur, Deepak; Crupi, John; Malks, Dan: Core J2EE Patterns: Best Practices and Design Strategies. , Prentice Hall, 2001


[IBM01] IBM corp.: WebSphere Performance Benchmark Sample (Trade 2 Application). 2002,

[SUN02e] Sun Microsystems: Design Patterns: Fast Lane Reader. 2002,
http://java.sun.com/blueprints/patterns/FastLaneReader.html

[BEG02] Begin, Clinton: JPetstore - A Pet Store from the Open Source Community. 2002,
http://www.ibatis.com/jpetstore/jpetstore.html
Alphabetical Index

activate method  6
Ant  27
Appender, log4j  28
Architecture, Petstore  21
Bean Managed Persistence, BMP  5
Buy stock script, Trade 3  18
Common Gateway Interface  3
COMPAS  1, 12, 16, 39
Container Managed Persistence, CMP  5
classical state  5
DAO, Data Access Object  9
Data Access Object, pattern  9
deployment descriptor  5
EJBCController  26
EJB specification  4
Enterprise JavaBeans  4
Entity beans  5
Event  25
Factory pattern  10
Fast Lane Reader  33
Front Controller, Pattern  7
HTMLAction  25
HttpServletRequest  25
Intercepting Filter, Pattern  8
J2EE Patterns  7
Java Messaging Service, JMS  6
JavaServer Pages  3
Layout, log4j  28
local client view  4
Log4j  27
Logger, log4j  28
MainServlet  25
Message Driven Beans  6
messaging service  6
Model-View-Controller, MVC  7, 21
OpenSTA  11, 17
Order Fulfillment Center  22
passivate method  6
Performance measurements  11
Petstore  21
Point to Point, JMS  6
Publish and Subscribe, JMS  6
queue, JMS  6
Quote stock script, Trade 3  18
remote client view  4
RequestProcessor  25p.
response time  20
RMI-IIOP  4
Sell stock script, Trade 3  18
Servlets  3
Session Beans  5
Session Facade, Pattern  9
SOAP  4, 14
stateful session bean  5
stateless session bean  6
StateMachine  26
Struts  26
think time  17
three-tier architecture  22
throughput  20
topic, JMS  6
Trade 2  12
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trade 3</td>
<td>14</td>
</tr>
<tr>
<td>TradeAppServlet, Trade 2</td>
<td>12</td>
</tr>
<tr>
<td>TradeScenarioServlet, Trade 2</td>
<td>12</td>
</tr>
<tr>
<td>Web Application Framework, WAF</td>
<td>25</td>
</tr>
<tr>
<td>WebController</td>
<td>26</td>
</tr>
<tr>
<td>Web Primitives, Trade 2</td>
<td>12</td>
</tr>
<tr>
<td>workload mix</td>
<td>15</td>
</tr>
</tbody>
</table>