TUHH
Technische Universität Hamburg-Harburg

Evaluation of Web Application Development Frameworks and Object-Relational Mappers: A Case Study

Student Project

Submitted by:

Sunay YALDIZ
Matriculation Number 22480
16.02.2004

TECHNICAL UNIVERSITY HAMBURG-HARBURG

GERMANY

Supervised by:

Prof. Dr. Joachim Schmidt
Software Systems Department
TECHNICAL UNIVERSITY HAMBURG-HARBURG
GERMANY

Acknowledgements

My special thanks go to Prof. Dr. Joachim W. Schmidt and research assistant Dipl.-Inf. Patrick Hupe at Software Systems Department of the Technical University of Hamburg-Harburg and Mr Uwe Langbecker at Germanischer Lloyd for providing the opportunity to work on this student project, for their essential and kind advice, encouragement and guidance throughout this student project work.

Contents

1. Introduction

Web application development is not an easy task if we consider web sites for a big company where dynamic pages are created. It covers much more than showing static html files, images. A real complex web application for an enterprise is just an interface for the overall tasks of the enterprise. Operations happening in the background for a web application include all kinds of enterprise applications, transactions, databases, Content Management Systems, etc. Operations vs. Systems do not match.

There are many existing web application development frameworks which ease the task of web application development. Struts [STRT03], Cocoon [CoCoN03] and OXF [OXF03] (Open XML Framework) are examples for these frameworks. These frameworks provide many libraries to enable the Web application developers to focus on the business processes implementation rather than focusing on programming language and environment related problems. In a way the frameworks suggest the separation of concerns for the development. The business objects and processes can be more concentrated on by the developers. The simple static pages' viewing of a web application is carried on a new platform where most of the data are dynamic, read from databases, where there exists high interaction of the system (web sites) with the clients (browsers). Without the frameworks the concept of dynamic page creation, development of content and business object-dense web sites is of course possible. But this necessitates very much programming which results in very error-prone sites that are developed in long time with high project budgets. The help of the frameworks are easier to realize at this point.

Object-relational mapping (O/R mapping) for persistence is mapping of the object-oriented world to relational databases where the information is held in relational structures. There is an impedance mismatch between the both worlds. But mapping is necessary for various reasons. Nowadays object-oriented languages like Java [Java03], C#, C++ are the mostly used languages by enterprises for software projects. The application and business data are modelled in an Object-oriented way. On the other hand, in relational databases the same data is expressed in relational tables which have columns and rows. In many projects the mapping from the Object-Oriented world to the relational world is manually coded in the application programs. This is a safe way but it is also very error-prone and requires a lot of hand-work. There is a need for automated mapping which abstracts from create, update and delete operations. There exist frameworks and tools for this automatic mapping task. Classes of the OO language are mapped to relational tables. Hibernate [HBNT03]; OJB (ObjectRelationalBridge) [OJB03] and Castor JDO [CJDO03] are some of the examples for these frameworks.

Germanischer Lloyd (GL) [GL03] is one of the leading classification societies worldwide. It supervises quality and safety mainly for ships and for maritime installations.
The GL Fleet Online is a service for customers of GL. GL Customers can view the status of their fleet and survey requests for the ships online. Fleet online makes it possible to get an access to current information for all GL classed vessels via internet. It supports the process of survey-planning, ordering and monitoring and provides the opportunity to manage customer generated items as well. GL Fleet Online service is currently implemented in the C programming language using CGI (Common Gateway Interface) [CGI03] technology. Sessions tracking and related online system requirements are manually coded. A move to current standard OO-Technologies and standards for web application development is planned

for the easier application development and maintenance. Java seems to be preferable as the development language and runtime environment.

New technologies for web development enable many features to provide for customers of GL. Mobile clients are used more and more frequently by the customers. There will be service provision for web-based and mobile clients.

1.1 Goal of the Work

In this student project an analysis of existing frameworks for web development will be done. Struts, Cocoon and OXF (Open XML Framework) web application development frameworks will be analysed and compared. The other main task of the work is the analysis of the O/R mapping tools. The O/R mapping frameworks Hibernate, OJB and Castor JDO will be analysed and compared. The main purpose for the analysis of the web application frameworks and O/R mapping frameworks is to implement the GL Fleet Online with current web-application technologies.

An analysis, design and prototypical implementation of the GL Fleet Online service with the chosen web development framework(s) and O/R mapping framework will be done. Not all the requirements identified for GL Fleet Online service will be implemented; the focus is on web application and O/R mapping tool evaluation.

1.2 Structure of the report

In chapter 2 a brief requirements analysis will be done for GL Fleet Online Service. In chapter 3, basic concepts about web development frameworks and O/R mapping frameworks will be explained. The web application development models, frameworks and O/R mapping frameworks will be explained. There will also be comparison of the analyzed web development frameworks and the analysed O/R mapping frameworks. Chapter 4 will be architecture for web applications on which the GL Fleet Online service will be realized will be defined. Short descriptions of the components with their purposes in the system will be given. In chapter 5, the implementation of the GL Fleet Online service with Struts, OXF, Hibernate and Castor XML will be explained. Selected implementation details will be given. Chapter 6 is the Conclusion section.

2. Requirements Analysis

In this chapter requirements analysis for GL Fleet Online service is done. The analysis is limited to requirements which are relevant for this student project. Before analysing the requirements, introductory information on Germanischer Lloyd is given.

2.1 Germanischer Lloyd

Germanischer Lloyd is one of the leading classification societies worldwide. As an independent and impartial society GL supervises quality and safety mainly for ships and for maritime installations. [GLv03]

GL Marine Services

- classification and certification of ships and offshore systems
- extensive engineering analysis and divisory services
- certification of marine related materials and components
- certification of quality management according to ISO 9000, ISO 14000 and ISM
- certification of maritime training according to STCW
- inspection and certification on behalf of 86 countries

Other GL Services

- evaluation and certification of pipelines, petrochemical plants and industrial installations
- analysis, testing and certification of materials and joining methods, failure analysis
- basic research, evaluation and certification of wind energy systems
- evaluation and certification of bridges and dams

For more information on GL services please refer to http://www.gl-group.com/.

2.2 Analysis

The Germanischer Lloyd (GL) is a classification society. It surveys ships, industrial plants and maritime installations with respect to safety. Surveys are done in regular intervals, normally every 5 years, but can be required more frequently. The time interval for surveys is provided by the ship classification company. The ship owner requests a survey for a ship from the classification society near the end of this period.

GL Fleet Online service has a main actor which takes part, the customer. Customer can login to the system, logoff from the system and inspect the status of the ships he has in the fleet.

Customer logs in GL Fleet Online Service using the username and password. If the login is successful, the customer can view the information about his ships. This information contains ship data and surveys data for these ships.

After logon, the customer can view the state of the fleet of the customer. The fleet has the information of the ships listed and survey data. The customer can choose a specific survey

request for viewing the details. After the survey request is chosen, the customer views the details for the chosen survey request of his ship.

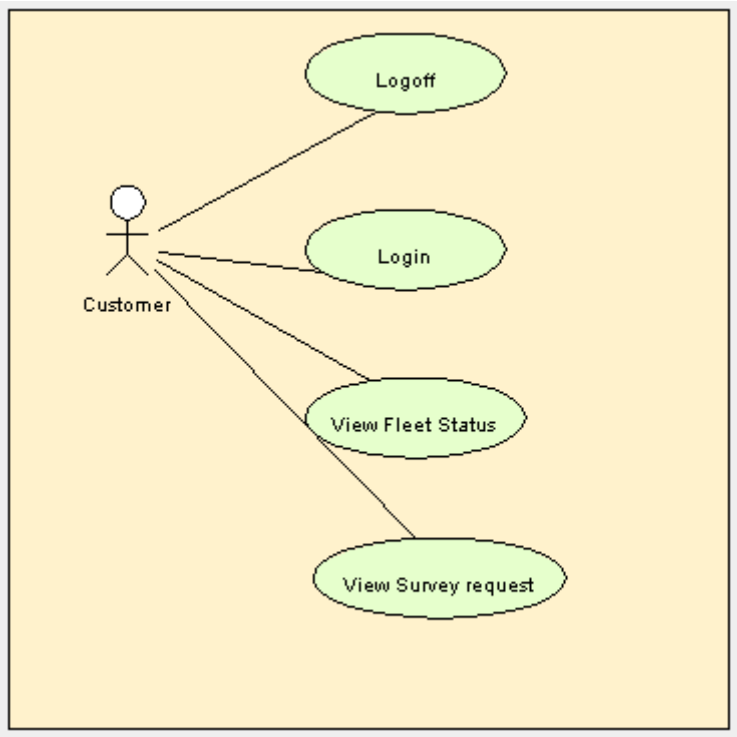

Figure 2-1 Use Cases for Fleet Online

When the customer finishes his work, he should logoff from the system. If he forgets to log off, after a timeout it will be automatically done. In Figure 2-1 the main use cases for the GL Fleet Online System can be seen.

Ships have survey requests. Survey requests for ships are needed for certification. All ships belonging to the customer need to be certified. The relationships between the customer, ship and survey and survey items can be seen in Figure 2-2 in conceptual class diagram (The cardinalities not shown are "1").

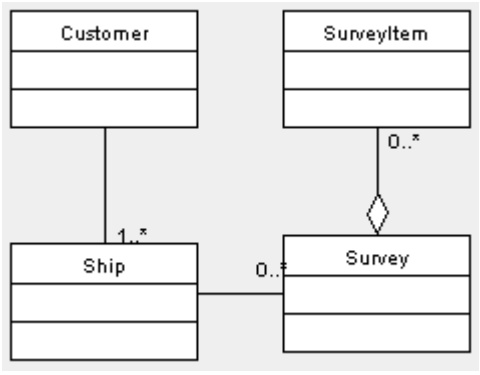

Figure 2-2 Conceptual Class Diagram for Fleet Online

The use cases analysed are the cases which will be implemented in this work. The use cases not seen like creation of new customers, survey data etc. also exist in Fleet Online system.

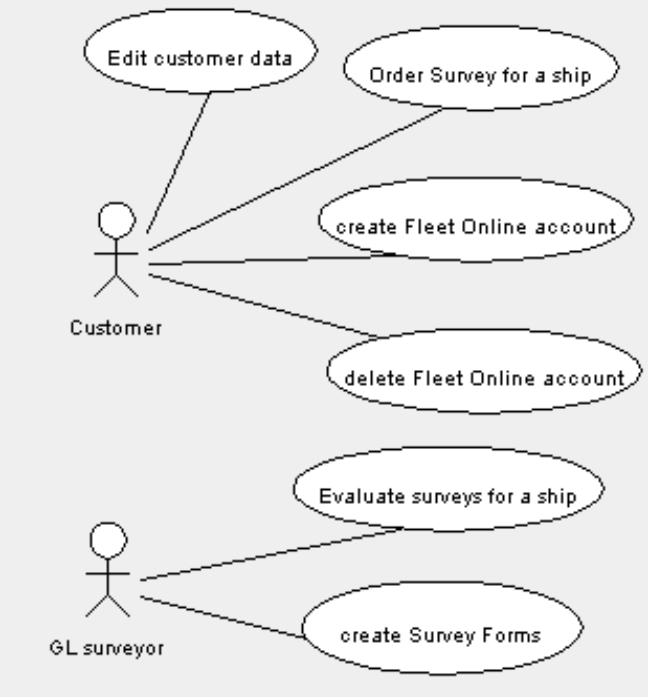

Figure 2-3 Use cases-unimplemented

The customers, ships and survey data reside in the main GL database Oracle. The surveys for ships are created on a rule-based system by GL surveyors.

The GL customers have a registration number. When they register as customer by specifying their ships information, customer information etc., the data are persisted. These data are then used by the GL Online Systems; Fleet Online is one of them. For accessing the ships, surveys data that belong to a customer, the customer should first get a user name and password from the Fleet Online System. So, the needs to create a Fleet Online System account. To be able to create an account, he gives the GL registration number and some more information.

The uses cases seen in Figure 2-3 are the unimplemented use cases. These use cases are necessary for understanding the overall Fleet Online System requirements analysed for this work. A customer as mentioned before creates a user account for Fleet Online based on his GL customer information like register number. When a customer does not need to use Internet access for Fleet Online, he can simply delete his web account. This does not remove his information in GL main database, only the Fleet Online related data are deleted. *Edit Customer Data* use case is also just about editing of the Fleet Online related customer data like web access properties, whether the customer wants to receive newsletters, etc.

An important use case is the *Order Survey for a Ship.* By giving the unique id of a ship, a customer can order a survey for this ship. The survey related data are entered by the customer.

Customer orders a survey and the evaluation of the survey is done in GL by surveyors. Ship data and related survey data are used to evaluate whether the rules for the survey hold. These rules are created also by GL surveyors as seen in use case *Create Survey Forms.*

These use cases are the main use cases of the Fleet Online System. The next section is about the existing Fleet Online System.

2.3 The Existing System

The existing web application GL Fleet Online service is for customers who have ships with specific survey requests. The details of the use cases for GL Fleet Online Service will be given in the next section.

GL Fleet Online Service is a server side web application to serve GL customers. As stated before in section 1.1, it is implemented with C and CGI (Common Gateway Interface) in an imperative way, not using Object oriented technologies. CGI for web applications is not an efficient way of web deployment and execution. For every request coming from the client browser a new operating system process is created. Process creation for Operating systems is a resource intensive and slow task. Especially the communication between the processes is hard and slow. Session tracking and related online system requirements are programmed “by hand”.

The existing database is Oracle [ORcL9i]. Persistence related functionalities are done by SQL commands in the C program.

All of the business data for GL Fleet Online System resides in RDBMS There is no business object model and it is a 2-tier architecture which is not state-of-art.

2.4 New Technologies for GL Fleet Online

A main task in this project is the prototypical implementation of GL Fleet Online Service with current technologies. The development language and runtime environment is chosen to be Java [Java03].

The Tomcat Servlet Container [TmCT03] is chosen as web application development and as servlet container runtime environment. Tomcat is a servlet container which controls the life cycle of servlets and the overall web application running under it.

Oracle RDBMS used as main productive database for GL enterprise applications will not be used for testing and development environment. The GL Fleet Online Service database content will be analysed and re-created in a test environment RDBMS. The chosen database is MySQL [MySQL03]. It is an open source database which has a very big user base, is tested very well, can be installed easily. It is even used in performance critical systems. Access to RDBMS was not available for this student project, only the database schema for GL Fleet Online Service of the existing system was provided. This necessitated use of a test database.

3. Concepts and Technologies

In this section concepts and technologies about web development frameworks and O/R mappers will be given and the selected frameworks will be analysed and compared. The analysis will result in an evaluation and choice of which framework and O/R mapper to use.

In the first section web application development frameworks are explained. For better understanding of the frameworks, an introductive section is given on a standard architecture for web applications. This architecture is for development of web applications in Java. There are two main architectures for web applications, model 1 and model 2. Model 2 is Model-view-controller architecture for server-side applications which are explained in more detail below. Another architecture analysed is the Model 2X which is an extension of Model 2 architecture.

In the second section, O/R mappers will be explained. The JDO [JDO03] standard and ODMG 3.0 [ODMG03] standards will be explained shortly. After this Hibernate, OJB and Castor JDO are explained.

3.1 Architecture for Web Applications

In this section 3-tier architecture for web applications will be described. The 3-tier architectures have the tiers: Presentation, Business logic and Database tier. The tiers will be shortly explained. The tiers can be seen in Figure 3-1.

The purpose of the Presentation tier (Layer) is the visualization of the contents like static and dynamic HTML [HTML03] pages. Presentation services provide the user interface to the system. The realization of the Presentation tier of a Java application is separated in two parts: Web Server for the static content and the Servlets, JSPs, EJB etc. technologies for the dynamic content creation. This tier presents the business data to be displayed.

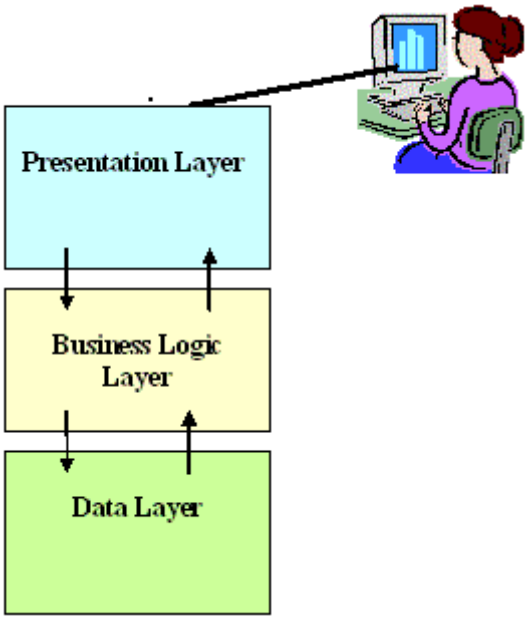

Figure 3-1 Tiers of 3-tier architectures

The Business Logic tier is the tier where the business process and Domain Objects modelling is done. The processes define the interfaces between the user of the system and the system. . In three-tier architecture, all user access to the data occurs through presentation services, which communicate with the business logic services.

The Database tier is the persistence tier. This tier provides the data on which the Business Logic tier work .The business objects are persisted for recovery failure situations. Main realization technologies for this tier are RDBMS, JDBC, and EJB etc.

Having 3-tier architecture for an application has many advantages as follows:

- Scalability and deployment flexibility - component roles are specialized, improving maintainability, networking, and I/O overheads.
- Component roles are clearly defined within a 3-tier framework. This provides a good basis for component-based development and reusability. Components in the business tier can be shared by any number of components in the presentation tier.
- Infrastructure independence is enhanced by the use of 3-tier architecture. This is because presentation and data access - areas that are often infrastructure-dependent are separated from the application's business logic.
- A specific set of skills is required for the development of each tier, so tiers can be developed independently of each other. For example, the thin presentation tier allows front-end experts to do their work without being affected by developments taking place in the business logic tier.

3. 2 Architectural Pattern: MVC

The MVC (Model-View-Controller) [Fowler03] architecture has its roots in Smalltalk, where it was originally applied to map the traditional input, processing, and output tasks to the graphical user interaction model. The MVC has three parts:

- Model

Model is the core of the application which maintains the state and data that the application represents. It knows nothing about the views representing this model.

- View

The view renders the contents of a model. It specifies how the model should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes. There may be several views representing the same model.

- Controller

The controller translates user interactions with the view into actions to be performed by the model. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in a Web application, they appear as GET and POST HTTP [HTTP03] requests. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

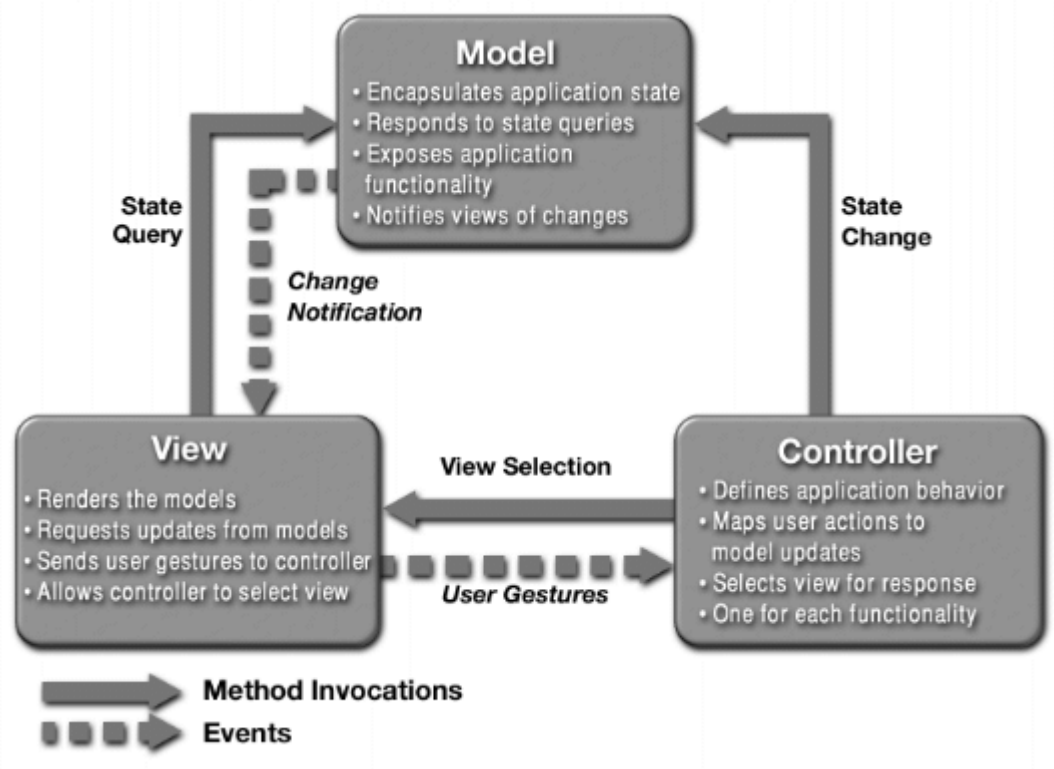

Figure 3-2 MVC architecture [SunBlue03]

The architectures to be explained in the next sections are based on MVC architecture. MVC architecture is a general architecture which can be applied to any application. But the models which will be explained soon are specific for server-side web applications. Specifically Model 2 is an MVC application for server-side web applications.

3.3 Java-based realizations of MVC: Model 1, Model 2, Model 2X

In this section architectural models which are realizations of MVC will be described; namely the models Model 1, Model 2 and Model 2X. Model 1 and Model 2 are models accepted by many users, they are defined by Sun according to how JSP are used in the web application. On the other hand, the Model 2X is not a model defined by Sun. It is not a model accepted and widely known by many users.

To understand the models some concepts of web development with Java shall be understood. They will be shortly explained in the following paragraphs.

A specification is developed by Sun Microsystems that defines how Java objects interact. An object that conforms to this specification is called a *JavaBean*.

Servlets are the Java platform technology of choice for extending and enhancing Web servers. Servlets provide a component-based, platform-independent method for building Web-based applications, without the performance limitations of CGI programs. And unlike proprietary server extension mechanisms (such as the Netscape Server API or Apache modules), servlets are server- and platform-independent. This leaves you free to select a "best of breed" strategy for your servers, platforms, and tools. [SRVT03]

Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. Servlets can also access a library of HTTP-specific calls and receive all the benefits of the mature Java language, including portability, performance, reusability, and crash protection. [SRVT03]

JavaServer Pages (JSP) technology enables Web developers and designers to rapidly develop and easily maintain, information-rich, dynamic Web pages that leverage existing business systems. As part of the Java technology family, JSP technology enables rapid development of Web-based applications that are platform independent. JSP technology separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content. [JSPv03]

JSP technology is an extension of the servlet technology created to support authoring of HTML and XML pages. It makes it easier to combine fixed or static template data with dynamic content. [SRVT03]

Enterprise JavaBeans (EJB) technology is the server-side component architecture for the Java 2 Platform, Enterprise Edition (J2EE) platform. EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology. [EJB03]

3.3.1 Model 1

Model 1 JSP page designs encapsulated the View and the Controller parts of the MVC pattern within the same JSP page (or pages), and the data underlying the application (the Model) was represented using JavaBeans. Model 1 consists of using JSP to extract data from the HTTP request parameters, applying the business logic implemented in JavaBeans or directly into the JSP, and then rendering the result in HTML output. In Model 1 view and controller are realized in JSP pages, two-tier is more coarse-grain. The concepts model, view and controller from the MVC pattern are not well separated in this model.

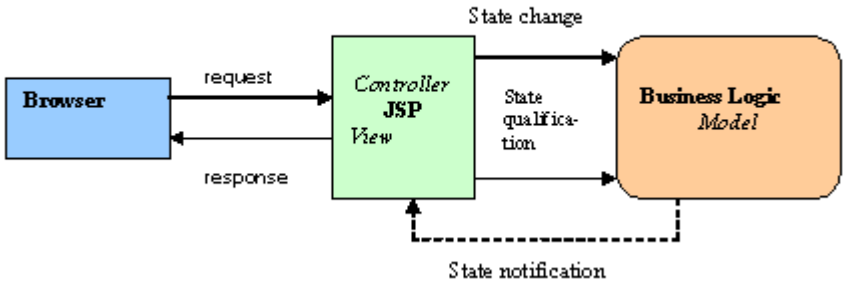

Figure 3-3 Model 1 components

The advantages of this model are that it is fast to implement, there is no overhead for dispatching requests and all views are in one JSP mixed. The disadvantages of model 1 are that there is no scalability and extensibility, there is strong coupling of responsibilities, and JSP page is a mixture of Java code and HTML which means the model and the view mixed.

3.3.2 Model 2

Model 2 is a server-side implementation of the Model View Controller architecture using Servlets and Java Server Pages (JSP). In Model 2, Servlets control the flow of the Web application and delegate the business logic to either JavaBeans or EJBs, while JSP pages take care of producing the HTML (view). This model has a much cleaner separation of business and presentation logic. This simplifies application development and maintenance.

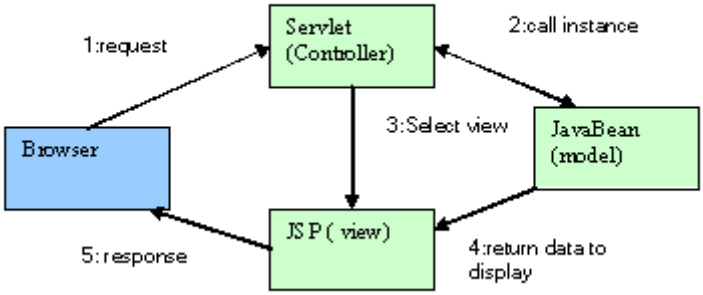

Figure 3-4 Model 2 components [OxfRef03]

3.3.3 Model 2X

Model 2X architecture is defined by Orbeon company [ORBE]. This model extends the Model 2 architecture. The only difference between Model 2 and Model 2x can be seen at the view layer. In Model 2X the views are created by applying XSL Transformations (XSLT) [XSLT03] stylesheets to XML documents instead of JSPs.

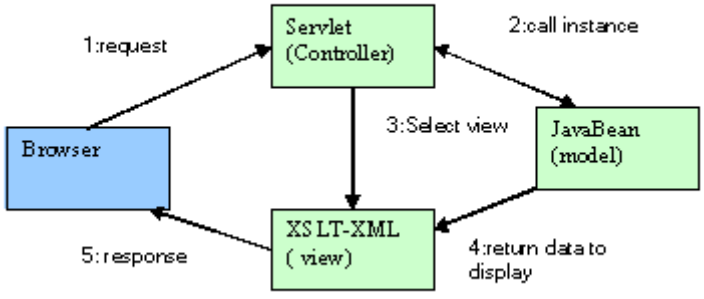

Figure 3-5 Model 2x [OxfRef03]

The difference between Model 2 and Model 2X can be seen in Figure 3-4 and Figure 3-5. As stated before, the view layer differs in the models.

The view layer of Model 2X enables this model to have very good separation of tiers in implementation. The business logic and the presentation are completely separated. Applying XSLT stylesheet to an XML document produces the presentation. In stylesheets there can not

be any business logic implementation. By this, the application programmer can not fall to trap of mixing business logic and presentation.

3.4 Web Development Frameworks

In the following subsections, the following frameworks for web application development will be described: Struts, OXF (Open XML Framework), and Cocoon and Struts-OXF integration framework.

The Struts-OXF integration framework which is an extension of Struts by OXF is provided by the company offering OXF. Normally integrating two different frameworks in a web application is either not possible or very difficult.

To understand the framework Struts-OXF integration one should also understand the web application architectures. This framework applies the architecture Model2X [ORBE].

3.4.1 Struts

Struts [STRTSv03] is a Model-View-Controller (MVC) based framework which is primarily based on servlets and JSP technology.

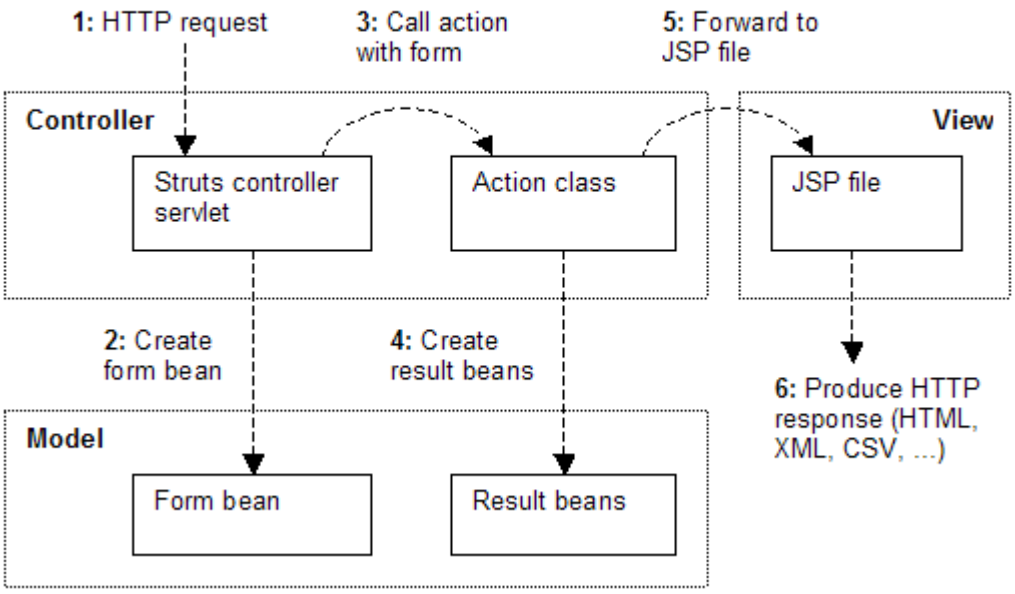

Figure 3-6 Struts elements and interaction [OxfRef03]

Struts is an implementation of Model 2 architecture. The Struts elements are explained below shortly. The Struts implementation of Model 2 layers, Model, View and Controller are to be seen in Figure 3-6 and Figure 3-7.

The Controller

Struts' main component is a generic controller servlet. The HTTP requests routed to Struts are handled first by controller. It interprets and dispatches all requests to individual actions implemented by the developer as subclasses of the Struts `Action` class. An XML file (*struts-config.xml*) that maps request URIs [W3CURI] to actions and form classes configures the controller servlet.

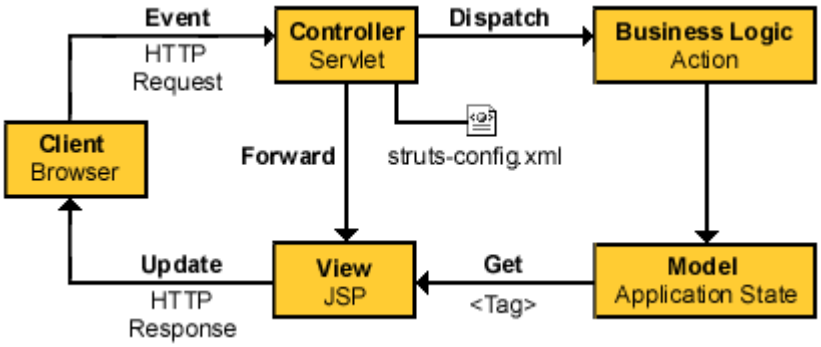

Figure 3-7 Struts Model 2-summarized [IBMSTR03]

At application start-up, the `ActionServlet` reads configuration file and initializes resources to map events to Struts `Action` classes. `ActionServlet` which is seen in Figure 3-8 is a standard implementation of the controller. It is the main dispatcher (controller). `ActionServlet` refers to the Controller part of the MVC implementation. The Controller creates and uses `Action`, an `ActionForm`, and `ActionForward`.

`Action` is a standard base class for business logic components and adapters; it is a wrapper of the business logic. The aim of `Action` class is to map `HttpServletRequests` to business logic. The existing actions, i.e. subclasses of Struts `Action` class, are mapped to logical names by request processor. `Action` implements the "Command Pattern" [GHJV95]: The execute() method is invoked for an incoming request, it can create response content directly. Typically it returns an `ActionForward` object to select the resource which prepares response that will be sent to client browser.

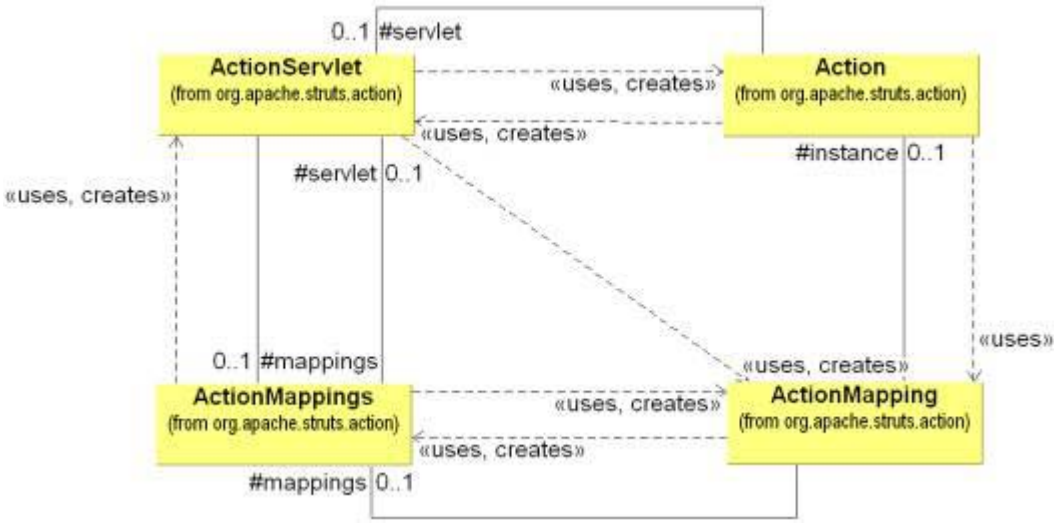

Figure 3-8 UML Diagram of the relationship of the Controller to the Model [IBMSTR03]

The Model

In Struts model layer, many implementation techniques to persist model entities can be used: JDBC, EJB, and O/R mapping tools with help of the `Action` class. `Action` is a business logic wrapper class which has references to the model elements used. The `Action` acquires information from the model and then exposes the information as request/session attributes. The components of view layer use these attributes for content rendition. It has access to all web related functionality like HTTPRequest, HTTPResponse classes as well as model beans.

The model takes the form of one or several JavaBeans. Those beans fall into three categories:

- *Form beans* hold request parameters. For instance, a form bean for a login page could have two attributes: login and password. Form beans extend the Struts `ActionForm` class seen in Figure 3-9. There are two types of Form Beans, `ActionForm` and `DynaActionForm`. `ActionForm` has a standard JavaBean design pattern. `ActionForm` maintains the session state for the Web application. `ActionForm` is an abstract class that is sub-classed for each representation of form input. `DynaActionForm` is a specialized subclass of ActionForm that allows the creation of form beans with dynamic sets of properties, without requiring the developer to create a Java class for each type of form bean. The definition of the property names and their types are done in *struts-config.xml*. In addition to properties, form beans define two standard methods: reset() and validate().The Reset() method resets form properties to initial value, and validate() method performs attribute-value validations.

- *Request beans* hold information needed to generate the HTML page. For example, in the case of a bank account statement page, a request bean would have properties with information about the account as well as a transaction bean collection detailing the most recent transactions to display.

- *Session beans* hold session information that persists between two HTTP requests of the same user session.

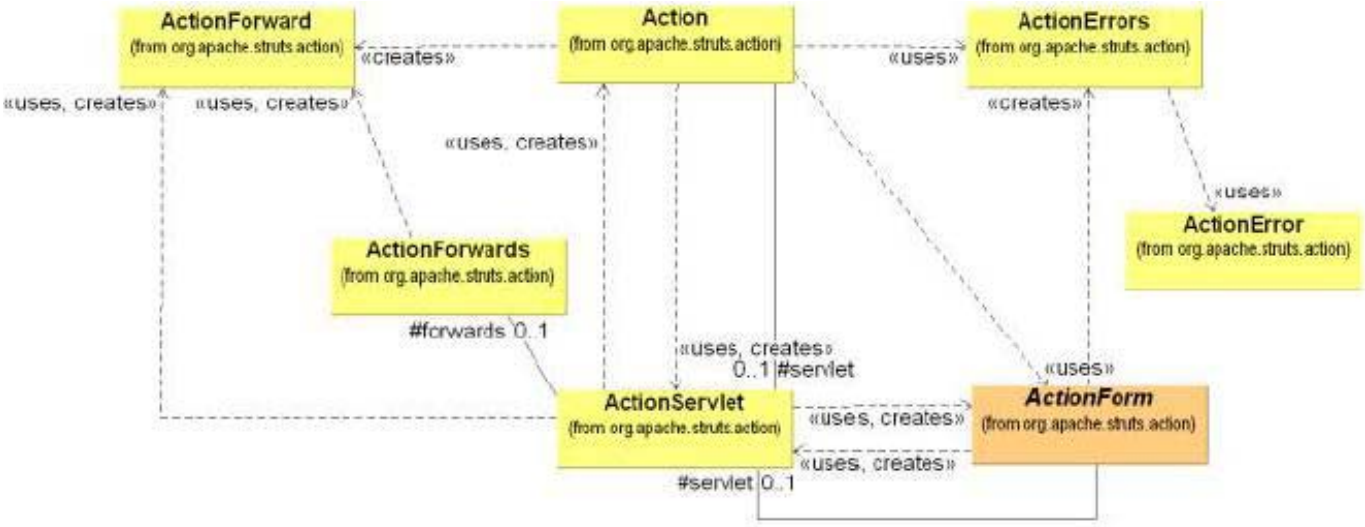

Figure 3-9 UML Diagram of the relationship of the Controller to the Model [IBMSTR03]

In Figure 3-9 a summarized UML class diagram of the `org.apache.struts.action` package is shown. It shows the minimal relationships among `ActionServlet` (Controller), `ActionForm` (Form State), and `Action` (Model Wrapper) classes.

The View

The controller realized as an `ActionServlet` forwards a request to a JSP implementing the view. The JSP can access the form, request, and session beans and outputs a result document (usually an HTML document) which is then sent to the client. Struts provides its own JSP Standard Tag Libraries (JSTL) [JSTL03] There are 4 JSP Custom Tag Libraries:

- struts-bean.tld – Fundamental bean manipulation and internationalization
- struts-html.tld – "Smart" HTML elements
- struts-logic.tld – Basic conditionals and iteration
- struts-template.tld – Basic layout management

Using the Struts tags in the JSPs helps avoiding use of Java code. Internationalization Support enables locale-specific applications.

3.4.2 OXF

OXF is an XML transformation framework based on J2EE [J2EE03] technologies. Initially planned for development of web applications, it now can be used for various applications including XML-based application data.

OXF is based on some fundamental concepts: XML processors and XML pipelines, the XML Pipeline Definition Language (XPL), the Web Application Controller (WAC) which are needed to build an OXF application. These will be explained in the next paragraphs.

XML Processors

XML Processors are components used to parse, transform, exchange, and present XML content. Many applications of XML processors exist.

Specification	***Implementation***	***Purpose***
XSLT	Apache Xalan, Saxon…	General XML transformation language. Can be used as "template language" to produce HTML pages.
W3C Schema	Sun Multi-Schema Validator	Checking XML documents against standard W3C schema.
XSL-FO	Apache FOP	Generating PDF documents from XSL-FO documents.
XForms	OXF XForms Processors	Form handling.

Table 1 XML Specifications

An XML processor is an XML component with XML inputs and XML outputs. An XSLT transformer is an example for an XML Processor. An XSLT processor transforms an XML input according to defined stylesheet into a new XML document.

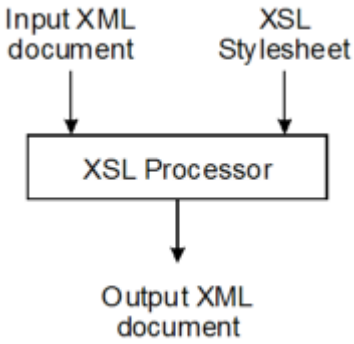

Figure 3-10 XSLT Transformer as XML Processor

OXF has its own XML Processor API. OXF encapsulates existing components' API (such as XSLT transformers). OXF has defined its own XML Processor API. New processors which implement this API can be written to create new processors.

XML Pipelines and XPL

OXF has its own XML elements defined used to combine several XML Processors, i.e. to create XML pipelines. In an XML pipeline, the XML Processors defined are applied sequentially. Consider such a pipeline which first executes an SQL query on a relational database using some SQL processor, and then transforms that output data to HTML table, and finally creates web page with a table and a header and a side menu. Figure 3-11 shows this pipeline.

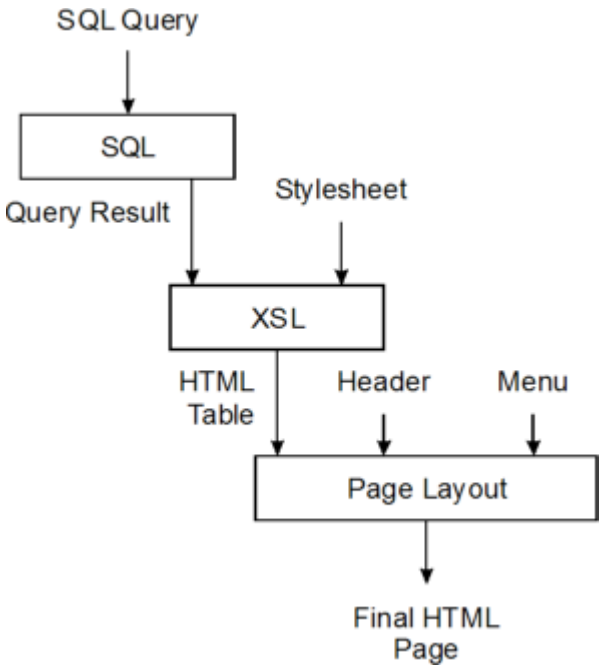

Figure 3-11 An XML Pipeline [OxfRef03]

Here there are 3 processors in the pipeline, one processor for SQL execution, one for XSL transformation and the third one for web page creation. The processors have XML inputs and outputs. All the elements must be in a namespace with the URI "http://www.orbeon.com/oxf/pipeline". <config> is the root element of a XPL document and

it defines zero or more input or output parameters to the pipeline with <param> parameters and the statements to be executed in the pipeline where a statement defines either a processor with its connections to other processors in the pipeline using <processor> , or a condition using <choose>. The <param> element defines the inputs and outputs of the pipeline. The inputs and outputs are for the pipeline, they do not need to be inputs/outputs of the processors in the pipeline. These elements have also various attributes for specific configurations which are omitted here. Table 2 shows the XPL definition for the pipeline seen in Figure 3-11.

```
<p:config xmlns:p="http://www.orbeon.com/oxf/pipeline">
<p:processor uri="oxf/processor/sql">
<p:input name="config">
 <sql:query xmlns:sql="http://orbeon.org/oxf/xml/sql">SELECT * FROM employees</sql:query>
 </p:input>
 <p:output name="data" id="employees"/>
 </p:processor>
<p:processor uri="oxf/processor/xslt">
 <p:input name="config">...</p:input>
 <p:input name="data" href="#employees"/>
 <p:output name="data" id="employees-table"/>
 </p:processor>
<p:processor uri="oxf/processor/aggregate">
 <p:input name="config">...</p:input>
 <p:input name="data-1" href="#employees-table"/>
 <p:input name="data-2" href="header.xml"/>
 <p:input name="data-3" href="sidebar.xml"/>
 <p:output name="data" id="html-page"/>
 </p:processor>
 </p:config>
```

Table 2 XPL definition for the SQL pipeline

Once XML pipelines are available, the presentation layer of an application can be instructed to use XML pipelines for implementing or integrating the business logic of the application. XML processors can do various tasks like accessing relational databases (SQL Processor), calling existing EJBs and Web Services (Delegation Processor) and controlling the application workflow (Web Application Controller). Custom processors can be also implemented using the *OXF Processor Java API* [OxfAPI03].

Web Application Controller

The OXF *web application controller* maps incoming user requests to individual pages and defines how each page is built out of a model and a view, following the model-view-controller (MVC) architectural pattern.

The Web Application Controller consists of a workflow engine which enables to define the entire site navigation logic. With this central page navigation logic, the pages can be developed independently from each other.

The Web Application Controller encourages separation of concerns in the applications between site logic, page logic (model in MVC), the page layout (view in MVC), and the site presentation. Site logic is the web application workflow which defines the navigation between the pages. Page logic defines the application logic objects and processes. Page layout defines how the information is displayed and presented to the user. Site Presentation covers the layout as well as look and feel common to all pages in the Web application or the Web site.

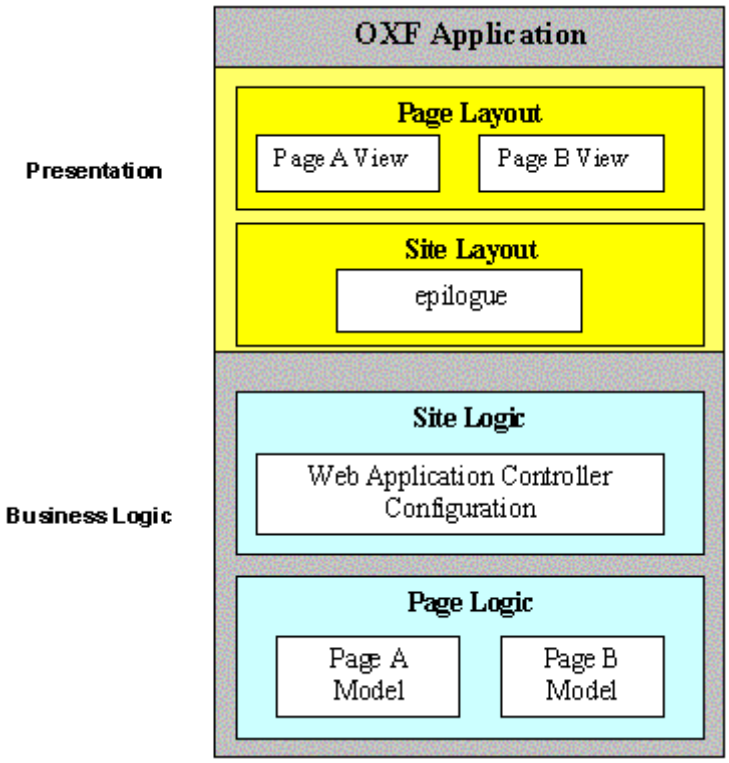

Figure 3-12 OXF: Presentation and Logic layers [OxfRef03]

As an example, for a ship news page, the site logic can retrieve the list of headlines and pass this information in an XML format to the view (page layout and presentation). The view produces an HTML page by creating a table with those headlines, potentially adding a logo at the top of the page, a copyright at the bottom, and so on.

OXF has a component called "epilogue" which allows defining a common look and feel for the page presentation. If epilogue feature is used by the application developer, the XML output for each page is used as input to epilogue which renders all pages with a common look and feel. The process is shown in Figure 3-13.

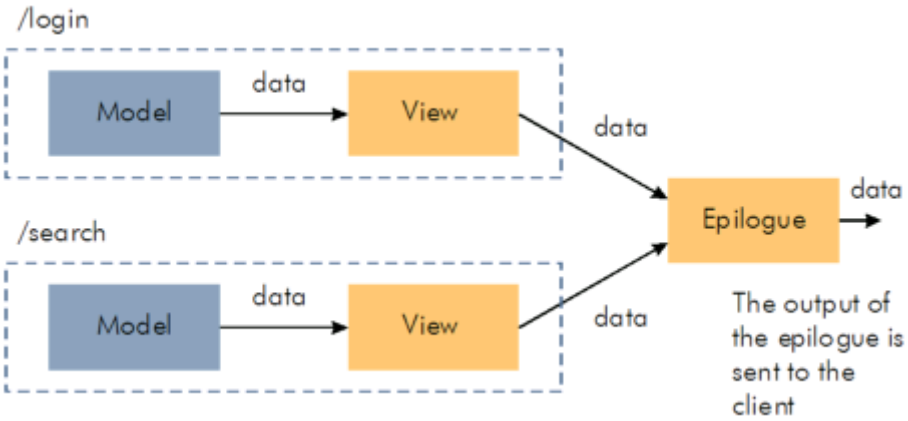

Figure 3-13 Epilogue in Controller [OxfRef03]

Finally the structure of the web application controller configuration file will be shown in Figure 3-14. The configuration file is an XML file composed of various elements as files, page and view. The files elements list the files that must be sent directly to the client, such as images or Cascading Style Sheet (CSS) [CSS03] files, i.e. the static content. The page elements declare pages or groups of similar pages and define the XForms [XForms03] model,

the MVC model, and the MVC view for each page. The view element defines properties that apply to all the pages [ORBE].

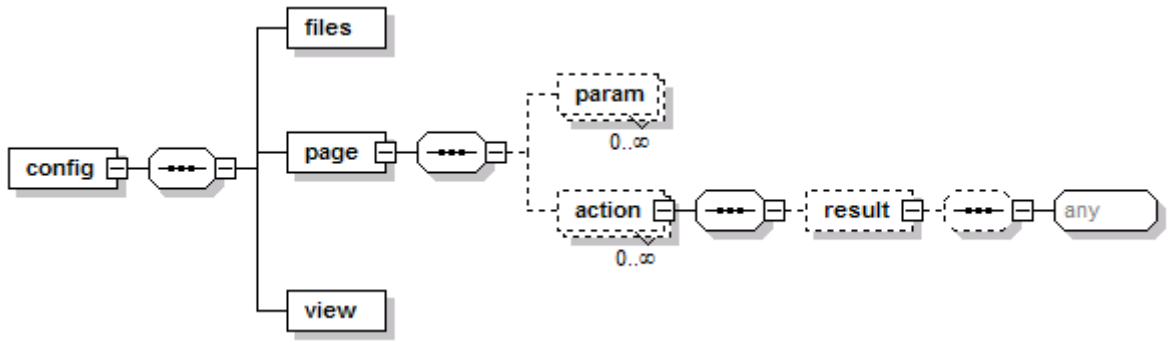

Figure 3-14 Web Application Controller configuration file element structure [OxfRef03]

An example for <config> element structure and <page> element can be seen in the following tables.

```
<config>
  <page path-info="/login" model="login/model.xpl" view="login/view.xml"/>
  <page path-info="/search" model="search/model.xpl" view="search/view.xml"/>
  <view epilogue="epilogue.xpl"/>
  </config>
```

Table 3 Example for <config> element

```
<page id="view-account" path-info="/view-account" xforms="view-account-form.xml" model="view-
account-get-balance.xpl" view="view-account-view.xsl">
 <action when="/amount != "" action="view-account-action.xpl">
    <result id="success" when="/success = 'true'" page="anything-else"/>
    <result id="failure" when="/success = 'false'" page="low-balance"/>
 </action>
</page>
```

Table 4 Example for <page> element

The XML Schema for the Web Application Controller file element structure can be found in [WAC03].

3.4.3 Cocoon

Apache Cocoon is a web development framework built around the concepts of separation of concerns and component-based web development. It can be also called “XML Publishing Engine” because of its nature easing the creation of XML-based output pages, i.e., views. The processing in any request-response cycle includes mainly XML documents.

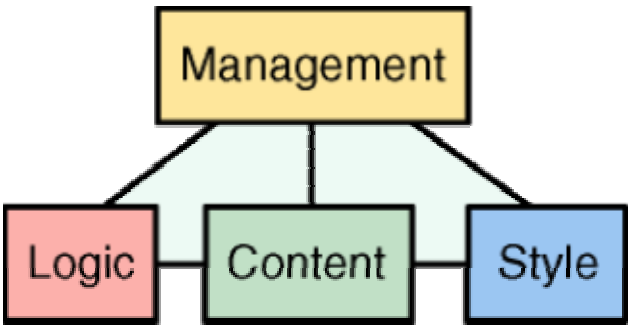

Figure 3-15 Separation of Concerns

Cocoon proves separation of concerns by component-wise application development as seen in Figure 3-15. The business logic, content and the style of the output pages can be completely separated by using separate components in the pipelines.

Cocoon implements these concepts by using 'component pipelines', in which each component on the pipeline specializes on a particular operation like generating XML content from the request or transforming the generated XML structure to another XML structure. In this way applications can be written by plugging together components into pipelines without any required programming.

3.4.3.1 Basic Mechanisms: Request-Response Cycle

When a request is sent by the client, it is dispatched to a Generator, which generates a first XML document. After this, XML transformations take place, documents can be transformed into another XML document, or any object type through Transformers (SVGs, PDFs). Aggregation of XML documents can take place through Aggregators, and lastly the rendering of output of aggregation is done through Serializers. Figure 3-16 shows the overall scenario for basic mechanisms.

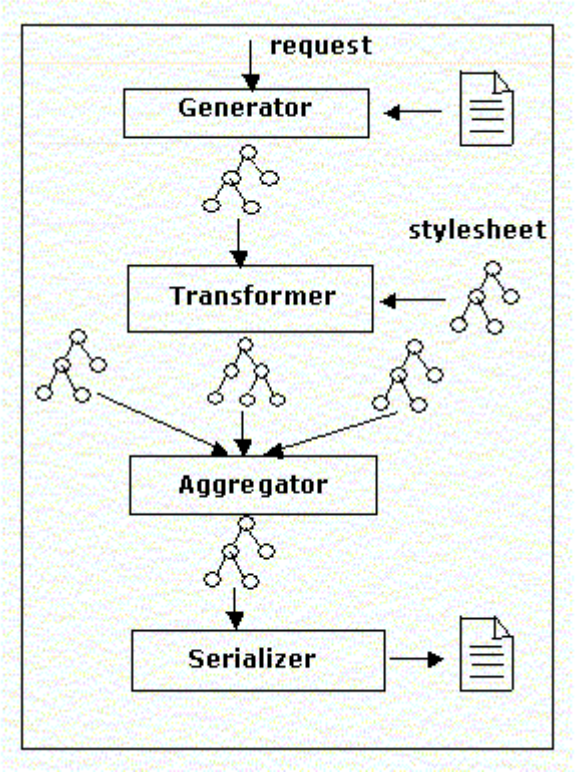

Figure 3-16 Cocoon Request-Response Cycle [COCO03]

This basic mechanism for XML-based web applications can be written as a pipeline defined in Cocoon, where the pipeline contains specific components for matching URIs, generating XML documents, transforming them, aggregating XML documents, and finally serializing to show the page.

Pipeline Dispatching and Invocation

The pipeline dispatching and invocation can be seen in UML Sequence Diagram in Figure 3-17. When the request from the client comes, Cocoon dispatches the request and the control passes to the SiteMap instance of Cocoon which is explained in more detail later. The SiteMap selects the corresponding pipeline and then the application control passes to the Cocoon pipeline which has all the application logic and data processing operations. The pipeline generates the output and it is sent to client.

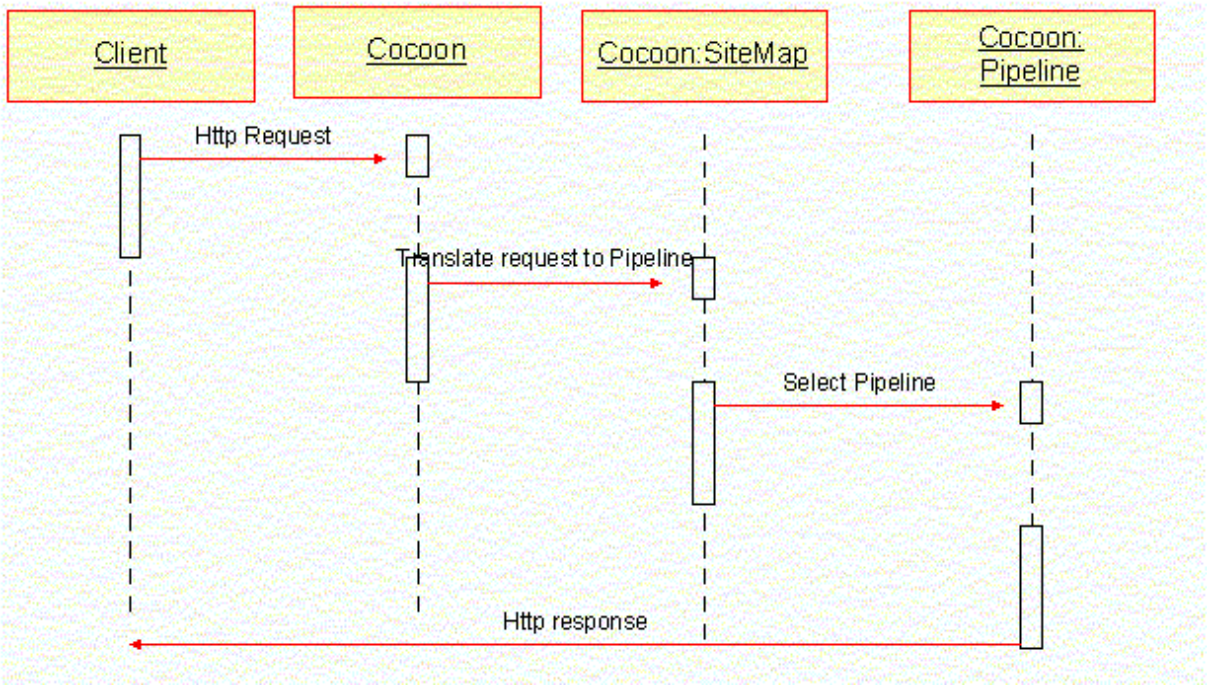

Figure 3-17 Sequence of Interactions[COCO03]

A simple sample pipeline is seen in Figure 3-18 .The request is given as input to a FileGenerator whose output is given to XSLT Transformer processor in the pipeline. The output of XSLT transformer is serialized into a HTML page by HTMLSerializer component. The response is sent to the client.

A minimal pipeline requires at least a generator and a serializer, but most pipelines have many transformers and serializers.

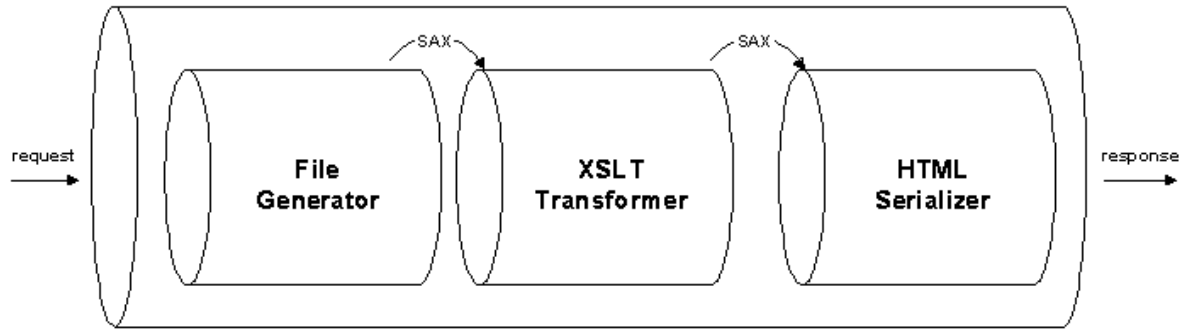

Figure 3-18 A simple sample pipeline [COCO03]

3.4.3.2 Architecture

Cocoon architecture has 4 main layers:

- *Core Cocoon* is composed of Avalon Framework(for logging, configuration , threading, context, etc), caching mechanism, pipeline handling, program generation, compilation, loading and execution, base classes for generation, transformation , serialization and components.
- Cocoon Components are generators, transformers, matchers, serializers, etc.
- Built-in Logicsheets are sitemap.xsl, xsp.xsl, esql.xsl, request.xsl, response.xsl, …
- Site specific configuration, components, logicsheets and content is the customization for the application.S

Figure 3-19 shows the simplified Cocoon architecture.

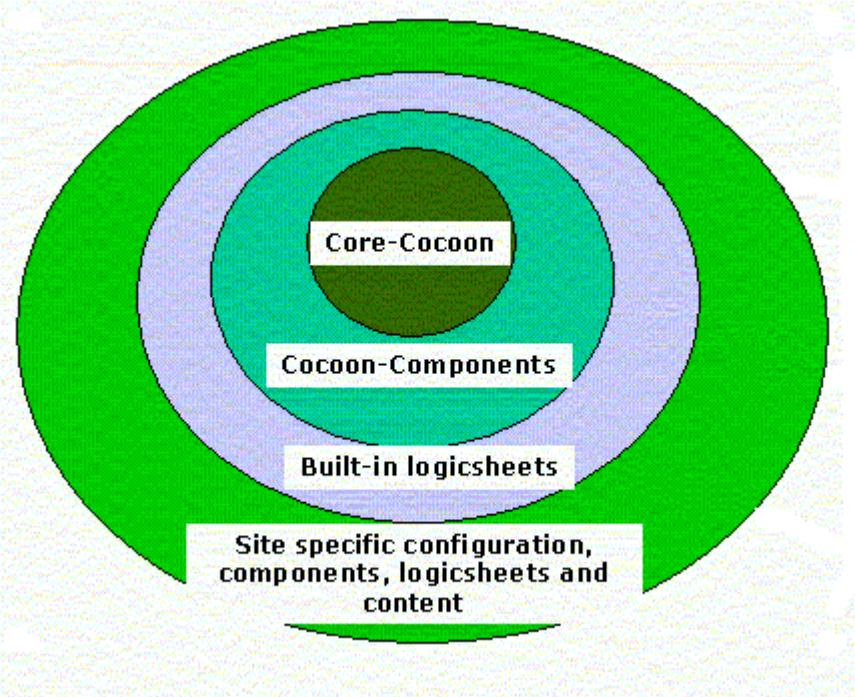

Figure 3-19 Cocoon Architecture [COCO03]

3.4.3.3 The Cocoon Abstraction of the Concepts:

This section explains how all the concepts mentioned above is implemented.

SiteMap

The sitemap component is the heart of a Cocoon application. The pipelines are defined in the sitemap configuration file. This is an XML file. It is composed of many elements as components, views, pipelines, matchers.

Matcher

A matcher attempts to match a URI with a specified pattern for dispatching the request to a specific processing pipeline. There are two kinds of predefined matchers in Cocoon: wildcard

matchers and regular expression matchers. The programmers can implement and add more matchers to Cocoon. Matchers help in choosing a specific pipeline for a URI group. In the Sitemap, the matchers are defined as <matchers> elements and names are given to matchers, and then they are used in pipelines.

Generator

A generator is used to create an XML structure from an input document like files, directories, streams, etc. Examples for predefined generators from Cocoon are file generator, directory generator, XSP generator, JSP generator, Request generator. Custom generators can be implemented by application developers. All generators used by web application are defined in the sitemap, and then they can be used in pipelines.

Transformer

A transformer is a component which maps an XML input document into another XML document, and returns it as output. Example transformers from Cocoon are the XSLT Transformer (the main transformer) which applies an XSL stylesheet to an input XML document, Log Transformer, SQL Transformer, and I18N Transformer. Like for the other components, programmers can implement and add custom transformers by registering them in configuration files.

Serializers

A Serializer is used to render an input XML structure into some other format (not necessarily XML) that is used primarily used for presentation. Examples for predefined serializers from Cocoon are: HTML Serializer, FOP Serializer, Text Serializer, and XML Serializer. Custom serializers can be added to Cocoon.

eXtensible Server Pages (XSPs)

An XSP page is an XML page where program logic and presentation can be implemented It should have language declaration which states in which programming language logic is implemented. It should also have namespace declaration. XSPs are like JSPs: Logic and view can be implemented together in an XSP page. An XSP is Cocoon way of mixing program logic and presentation necessary for simpler projects. The logic and the presentation code can be together in an XSP file or logic can be defined in logic stylesheets and later on imported to XSPs. An XSP page is used by a generator to generate an XML document. XSPs enable dynamic content generation in Cocoon. XSPs can be compared with JSPs.

Logicsheets

A logicsheet is an XSL stylesheet which is the primary mechanism to add program logic to XSPs. Logicsheets are used by the generator components to transform XML structures before generating program. The logicsheets can be packaged as a reusable tag library. Logicsheets enable effective separation of content, logic and management. Cocoon has some built-in logicsheets as request.xsl, response.xsl , session.xsl which are used to easily access the request, response and session in logicsheets.

A sample pipeline in sitemap.xmap is shown in Table 5. In this pipeline there are two main match patterns. In the first one, when there is a request for hello.html by the client, the generator component generates the page from the XML document specified by <map:generate> element. The output to be sent to client is serialized to html.

The second match pattern in the pipeline are for image loading. When the client requests an image with relative URI “images/*.png”, the image is read from the relative path “resources/images*.png” and sent to client.

```
<map:pipeline>
<map:match pattern=”hello.html”>
        <map:generate src=”docs/samples/hello-page.xml”/>
        <map:serialize type=”html”/>
  </map:match>

  <map:match pattern=”images/**.png”>
        <map:read src=”resources/images{1}.png” mime-type=”image/png”/>
  </map:match>
</map:pipeline>
```

Table 5 Sample pipeline definition in Sitemap.xmap

For more information on Cocoon Framework, see [COCO03].

3.4.4 Struts-OXF Integration

With the explanations of OXF and Struts given, the integration framework Struts-OXF integration will now be explained. OXF can be used in conjunction with Struts in a Struts-based application. The resulting application has Model 2X. Model 2 can also applied in some web applications if JSPs are used with XPLs (xml pipelines) as views.

In the integrated framework, Struts controller servlet `ActionServlet` is the controller of the web application. The requests are dispatched by Struts. OXF enables addition of xml pipelines and processing of those pipelines. When a request is mapped to an .xpl file, i.e. a pipeline, then OXF does the processing and produces the view, i.e., output page.

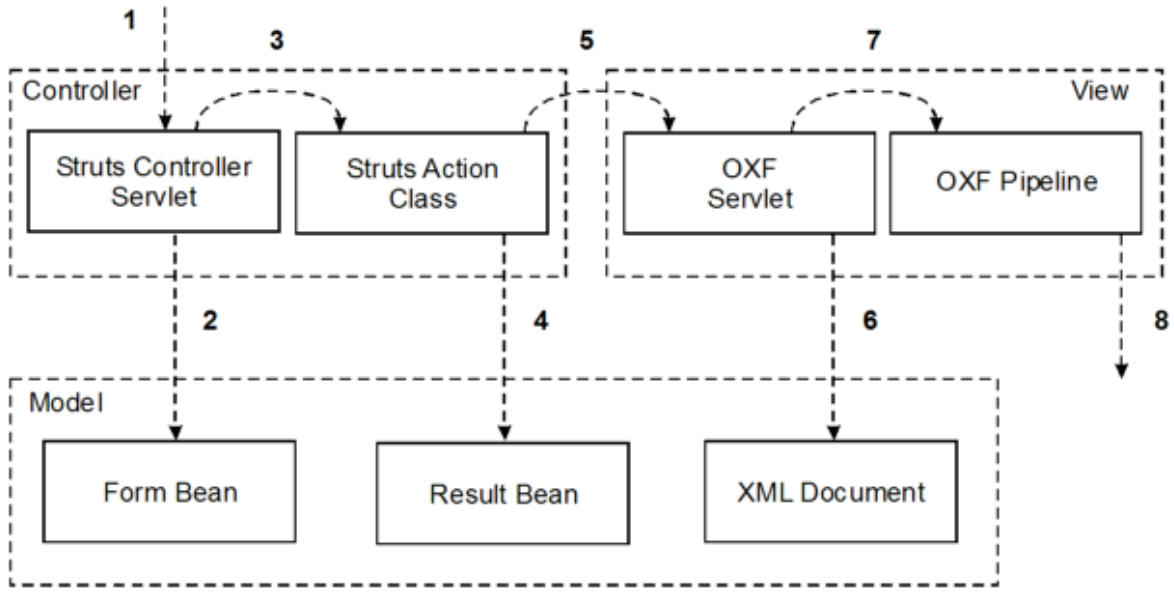

Figure 3-20 Control Flow of Request-Response cycle in a Struts-OXF application [ORBE]

Figure 3-20 shows the control flow where there are no JSPs as view, only OXF pipeline outputs producing the output pages, i.e. views. The Struts controller is responsible for business logic and dispatching to XML pipeline whenever necessary.

A Struts-OXF Web Application contains at least two servlets: the Struts controller servlet and the OXF processor servlet. The former is the Struts-based servlet, serving all URI, while the latter is bound to xpl files and other resources served by OXF (images, CSS, etc.). Which servlet serves which resources is defined in the web application descriptor. (*web.xml*)

As shown in Figure 3-20, the Struts controller interprets the request and creates form beans containing the request parameters. It then instantiates and calls the appropriate action class. The result bean is then set in the request, and the request is forwarded to the OXF processor. Struts selects the pages to forward just like for JSP forwards. In the following example, requests for "/hello" relative path are dispatched to first the HelloAction class, and then there is a forward to "/hello.xpl" relative path invoking the OXF pipeline which is handled by OXF servlet.

```
<action path="/hello" type="org.orbeon.oxf.struts.examples.hello.HelloAction" name="hello">
        <forward name="success" path="/hello.xpl"/>
</action>
```

OXF has a BeanGenerator component which enables most of the data transfer between Struts and OXF in the web application. In the OXF pipeline, the BeanGenerator component can access the beans in the request or session. It is used to serialize these beans into XML [ORBE].

3.4.5 Comparison of the inspected frameworks

The comparison of the inspected frameworks is based on the following criteria: Support, architecture/separation of tiers, stability, performance and ease of development.

Support

OXF has a small user community, the developer company ORBEON gives support when necessary. Cocoon has a big user community, and there are good tutorials and documentation available. Struts has a large and helpful user community with a big mailing list, very useful tutorials.

Separation of tiers

In OXF, separation of tiers is very good. OXF and Cocoon are MVC-based frameworks. In Struts it is also good, but there is still danger of mixing business logic and presentation in JSPs. In JSPs application programmers can implement both logic (Java code) and presentation (HTML and Tag Libraries).

Stability and Robustness

OXF is a still maturing technology; there are some bugs and missing features. Cocoon has a lot of stable features which makes it in total stable. Struts is on the other hand very well tested and stable framework.

Performance

OXF performance is slow as objects are not cached yet, but other than that, the performance is

good. There is not much documentation of Cocoon performance, caching etc. But both of the frameworks OXF and Cocoon are not as good as Struts in performance because of XML processing overhead. The fact that all data are represented in XML brings the need of many XML transformations and processing steps which is a major performance drawback.
Struts has very good performance, since it does not have XSLT processing, and JSPs are precompiled with every change.

Ease of Development

OXF can be difficult to use due to incomplete documentation and sometimes poor error messages, but there is a very good support from Orbeon Company. The difficulty of creating own XML processors is a drawback, as no Java API exists for writing custom processors. It is a framework with an XML core, so those who are not very much experienced with XML processing may have difficulty in learning.

Cocoon has very good separation of tiers, but complexity of the framework and complexity of sitemap and sub sitemaps navigation (web application controller and pipeline definitions in sitemaps) can take time to learn.

Struts is easy to learn and to use when the basics of servlets and web applications are known. The flexibility of Java programming in model (Action classes) makes it very suitable for big projects. Also it is simple to configure Struts as it uses the standard web application configuration mechanism. In Struts, JSP tag library is complex.

3.4.6 Selection of framework

If a project does not require the use of XML data, then Struts is the best choice with all of the available documentation, user base, and tutorials. For the projects where some of the data are represented as XML and some views are created by XML and XSLT processing, the choice is the Struts-OXF integration framework. There one has all the flexibilities Struts give for the development and whenever one needs to produce views from XML content, can be delegated to the OXF processor. That is why for this work Struts in conjunction with OXF is chosen.

3.5 Object-relational (O/R) Mapping

A major portion of the development of an enterprise application involves the creation and maintenance of the persistence layer used to store and retrieve objects from the database of choice. Many organizations resort to creating home-grown, often error-prone, persistence mechanisms. If changes are made to the underlying database schema, it can be expensive to propagate those changes to the rest of the application. There is a need to fill this gap, providing an easy-to-use and powerful object-relational persistence framework for applications. [HIBEa1]

Object-relational persistence and query tools are frameworks or libraries for Object to Relational mapping. They offer a higher level API compared to JDBC [JDBC03] for persistence and querying.

Object-relational mapping is a persistence mechanism which connects objects of an object-oriented layer to data stored in a relational database.

Using an object-relational mapping tool/framework allows application of object-oriented analysis, design and programming techniques by hiding the details of RDBMS from the programmer.

There are commercial and Open Source object-relational mapping tools (ORMs) that comply with the ODMG 3.0 and JDO standard APIs for object persistence. Programming against such a mapping API is not different from programming against an object-oriented database.

In the following sections three of the ORM tools/frameworks will be described, they are free license and open source tools/frameworks. They are compared and evaluated and one will be chosen for this work.

3.5.1 O/R Mapping Standards

Sun JDO Standard

The Java Data Objects (JDO) [JDO03] API is a standard interface-based Java model abstraction of persistence, developed as Java Specification Request 12 under the auspices of the Java Community Process. Application programmers use JDO to directly store their Java domain model instances into the persistent store (database) [Fowler03].

ODMG Standard

The Object Data Management Group (ODMG) Binding is a standard for O/R mapping of data objects.

There are several differences and similarities between JDO and ODMG 3.0 standards, but they will not be described here. These standards will be used in comparing the persistence tools/frameworks in next sections. Conforming to ODMG and/or JDO standards is a good criterion for comparison.

3.5.2 O/R Mapping Frameworks

In this section some O/R mapping frameworks will be analysed and compared. They are Hibernate, OJB and Castor JDO.

3.5.2.1 Hibernate

Hibernate is an O/R persistence and query framework for Java. In addition to persisting objects, Hibernate provides its own rich query language to retrieve objects from the database, as well as an efficient caching layer and Java Management Extensions (JMX) support. Hibernate provides support for collections and object relations, as well as composite types. User-defined data types and dynamic beans are also supported. [HibeRef03]

Hibernate can be used in standalone applications or it can be used in web application under servlet containers. The second approach simplifies some of the configurations as the servlet container takes care of them.

In the following subsections, Hibernate's architecture will be explained first, and then a sample mapping with Hibernate will be shown.

3.5.2.1.1 Architecture

In an application interacting with the database, Hibernate is the layer between the application and the database. Hibernate provides the application automated persistence of the application objects. For this Hibernate uses properties file where database, connection, etc. properties are given to Hibernate by the application programmer. The XML mapping files are used for mapping of the database tables to Java classes. Hibernate uses these mapping files for persisting the Java objects to database tables.

For providing persistence services to the application, Hibernate uses hibernate.properties and XML mapping files. The application has an abstraction of the database, for object saving, creation and queries. The properties file is a file for specifying Hibernate specific properties. XML mapping files are mappings for persistent objects to database schemas. The class-table and class field-table field mappings are done in this file.

Hibernate supports several approaches for runtime architecture.
One approach is that the application itself provides its own connections and the transactions. Hibernate will be used for persistence and query services. Not all the features provided by Hibernate are used by the application programmer in this approach. Figure 3-22 shows this approach. SessionFactory and Session are Hibernate classes.

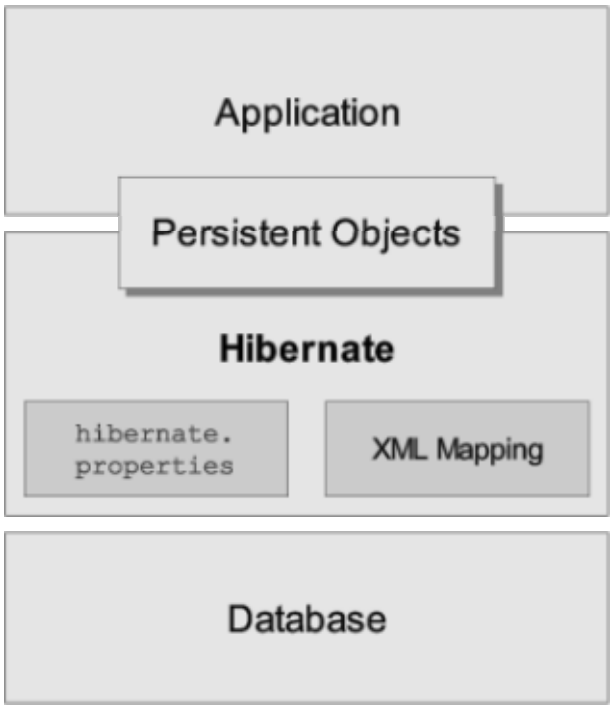

Figure 3-21 Hibernate Architecture [HibeRef03]

In the second approach seen in Figure 3-23, Hibernate does all of the work, from the connection to transactions. The application does not manage the database connections and transactions, Hibernate abstracts all of them. Here more classes from Hibernate API are used by the application: In this approach, the abstractions of database connections, transactions, and sessions are done by Hibernate classes `SessionFactory`, `TransactionFactory`, `Session`, `Transaction`, and `ConnectionProvider`.

The application uses the Hibernate framework components. Hibernate uses Java Naming and Directory Interface (JNDI) [JnDI03], JDBC, Java Transaction API (JTA) [JtA03] to do those abstractions for the application.

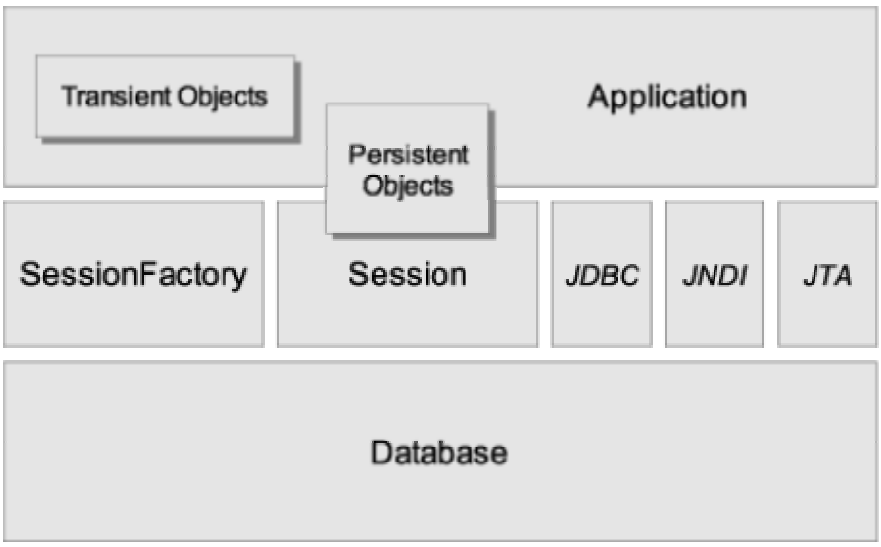

Figure 3-22 Hibernate-Light Architecture [HibeRef03]

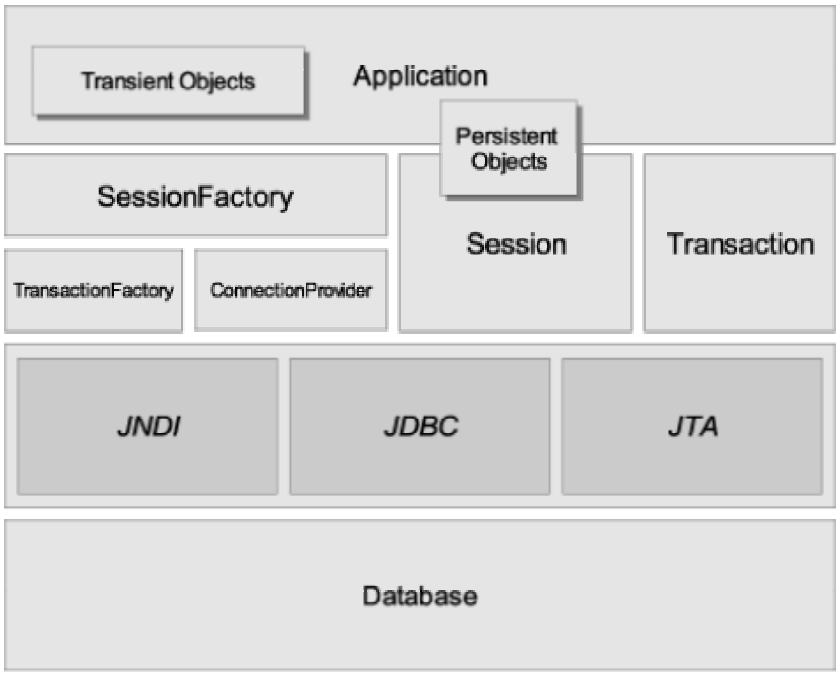

Figure 3-23 Hibernate-full architecture [HibeRef03]

The elements in Hibernate Architecture Figures

In the above figures Figure 3-22 and Figure 3-23 showing Hibernate architecture, some elements need to be explained more detailed. Some of them are Hibernate classes, `SessionFactory, Session, TransactionFactory`, and `Transaction.`

`SessionFactory (net.sf.hibernate.SessionFactory)` is a threadsafe and immutable cache of compiled mappings. It is used to create Session objects. It is also a client for ConnectionProvider class.

- `Session (net.sf.hibernate.Session)` is a short-lived, single-threaded object representing a session between the application and the database. It wraps a JDBC connection. It is a `Transaction` object creation factory. It keeps the persistent objects as cache.

- `Transaction (net.sf.hibernate.Transaction)` is also single-threaded and short-lived like Session. It is used by the application to specify atomic units of work. It abstracts application from underlying JDBC, JTA or CORBA transaction. A session may have several transactions living.

- `ConnectionProvider(net.sf.hibernate.connection.Connection Provider)` is a factory for JDBC connections. It can pool connections. It abstracts the application from the `DataSource` and `DriverManager` details for the connection.

- `TransactionFactory (net.sf.hibernate.TransactionFactory)` is a factory for Transaction instances.

In the first approach where the application takes care of connections to data sources, the classes `Transaction, TransactionFactory, ConnectionProvider` are not used. In case of second approach, Hibernate abstracts the connection and transaction details with those classes. The application uses JTA and JDBC.

3.5.2.1.2 An example for O/R mapping with Hibernate

The objects to be persisted are defined in a mapping document, which serves to describe the persistent fields and associations, as well as any subclasses or proxies of the persistent object. The mapping documents are compiled at application start-up time and provide the framework with necessary information for a class. Additionally, they are used in support operations, such as generating the database schema or creating stub Java source files. A `SessionFactory` is created from the compiled collection of mapping documents. The `SessionFactory` provides the mechanism for managing persistent classes, the `Session` interface. The `Session` class provides the interface between the persistent data store and the application. The `Session` interface wraps a JDBC connection, which can be user-managed or controlled by Hibernate, and is only intended to be used by a single application thread, then closed and discarded.

The examples are taken from [HIBEa1].

The class diagram seen in Figure 3-24 will be persisted to the tables seen in Figure 3-25. The class Team will be mapped to the table teams and similarly the class Player to the table players. The keys of the tables will be generated automatically by Hibernate when the objects are saved to database. The algorithm chosen for generation of keys uses an extra table, hibernate_unique_key.

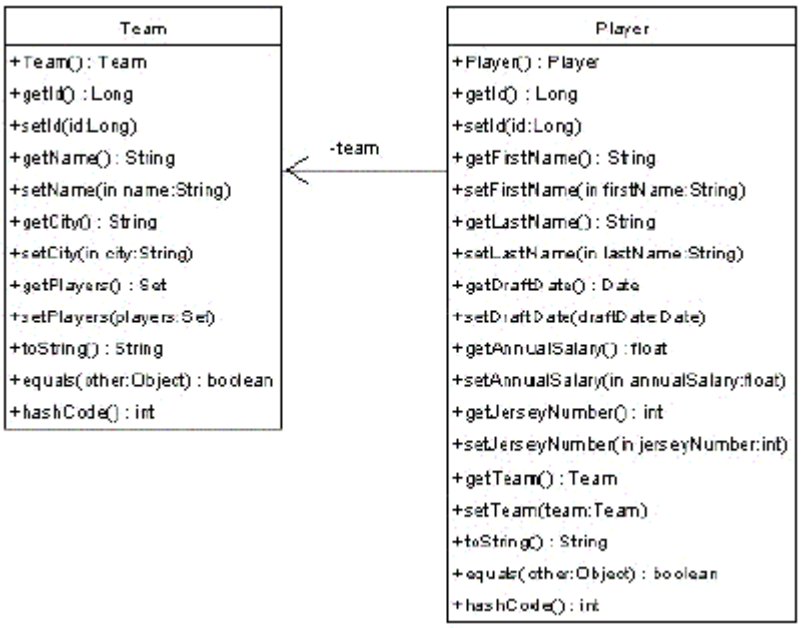

Figure 3-24 Class diagrams for Team and Player [HIBEa1]

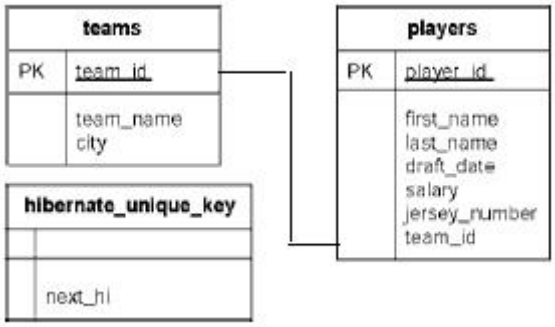

Figure 3-25 Related Database schema [HIBEa1]

The mapping files are clear enough in most respects. Some parts to explain are <id> element, <generator> element. <id> element defines the key column for the tables. Every object of the mapped classes should have unique ids. The <generator> element defines the key generation algorithm. One can use database specific key generators or write own key generators implementing generator interface, use the generators provided by Hibernate. There is also an *assigned* option where the id fields of the objects should be set in the application program.

In the mapping file, the root element is <hibernate-mapping> element. The element <class> defines the mapping for a specific Java class. The corresponding table in the database is provided in the specific attributes. Later on the fields of the Java class are mapped to columns in the corresponding table.

Although in the example given in Figure 3-24 and Figure 3-25 the classes do not have any collection fields, it is also possible to map in Hibernate various collection types like arrays, lists, maps. Relationships between classes like *one-to-one, one-to-many, many-to-many* can be also expressed in the mapping files.

```
<hibernate-mapping>
  <class name="example.Team" table="teams">
        <id name="id" column="team_id" type="long" unsaved-
value="null">
           <generator class="hilo"/>
        </id>
        <property name="name" column="team_name" type="string"
                        length="15" not-null="true"/>
        <property name="city" column="city" type="string"
                        length="15" not-null="true"/>
        <set name="players" cascade="all" inverse="true" lazy="true">
                        <key column="team_id"/>
                        <one-to-many class="example.Player"/>
        </set>
  </class>
</hibernate-mapping>
```

Table 6 Hibernate mapping file for example.Team [HIBEa1]

The tables Table 6 and Table 7 show the mappings for the `example.Team` and `example.Player` classes. The fields `id`, `name` and `city` of `example.Team` are mapped to team_id, team_name and city columns of the teams table. The team_id field in players table references the team_id column of teams table. Persistence of players field uses the team_id column in example.Player. With the players persisted to players table, the team's id to which this player belongs is also persisted. The 1: n relationship between the teams and players are established in the database schema by the team_id foreign key in players table.

```
<hibernate-mapping>
  <class name="example.Player" table="players">
        <id name="id" column="player_id" type="long" unsaved-value="null">
          <generator class="hilo"/>
        </id>
        <property name="firstName" column="first_name" type="string"
                        length="12" not-null="true"/>
        <property name="lastName" column="last_name" type="string"
                        length="15" not-null="true"/>
        <property name="draftDate" column="draft_date" type="date"/>
        <property name="annualSalary" column="salary" type="float"/>
        <property name="jerseyNumber" column="jersey_number" type="integer"
                        length="2" not-null="true"/>
        <many-to-one name="team" class="example.Team" column="team_id"/>

  </class>
</hibernate-mapping>
```

Table 7 Hibernate mapping for example.Player [HIBEa1]

The fields `id`, `firstName`, `lastName`, `draftDate`, `annualSalary` and `jerseyNumber` of the `example.Player` class are mapped to columns player_id, first_name, last_name, draft_date, salary and jersey_number columns in players table. As stated the team_id in this table references the team_id in teams table. The field team is persisted by help of this column.

Creating, Updating and Querying Persistent Objects

After the mappings are written Hibernate system initializes the mappings, in the application programs persistent Java objects for the mapped classes can be automatically created, updated, deleted or queried.

In the following code snippet in Table 8 the creation of a team object and saving it to underlying database can be seen. The created instance should be saved in a Hibernate session to insert it to database table.

```
Team team = new Team();
team.setCity("Detroit");
team.setName("Pistons");

//add a player to the team.
Player player = new Player();
player.setFirstName("Chauncey");
player.setLastName("Billups");
player.setJerseyNumber(1);
player.setAnnualSalary(4000000f);
Set players = new HashSet();
players.add(player);

team.setPlayers(players);
/open a session and save the team
Session session = SessionFactory.openSession();
Session.saveOrUpdate(team);
```

Table 8 Creating and persisting objects [HIBEa1]

Similarly as seen in Table 9, objects from database can be automatically loaded in the application program; Hibernate creates the necessary object(s) for the corresponding row(s) in the table. In the program, the developer has the object reference provided by Hibernate.

```
//method 1: loading a persistent instance
Session session = SessionFactory.createSession();
Player player = session.load(Player.class, playerId);

//method 2: loading the Player's state
Player player = new Player();
session.load(player,playerId);
```

Table 9 Loading objects from RDBMS with Hibernate [HIBEa1]

Deleting persistent objects from database are also possible as seen in the next code snippet.

```
//method 1: deleting the Player loaded
session.delete(player);

//method 2: deleting all of the Players with a
//salary greater than 4 million
session.delete("from player in class example.Player where
player.annualSalary > 4000000");
```

Table 10 Deleting objects from RDBMS with Hibernate[HIBEa1]

3.5.2.2 OJB (ObjectRelationalBridge)

ObjectRelationalBridge (OJB) [OJB03] is a mapping framework that allows transparent persistence for Java Objects against relational databases.

OJB is an open source Apache Licensed O/R mapping framework. It provides persistence for Java classes.

Architecture

The layers of OJB in an application architecture containing the backends can be seen in Figure 3-26. Currently some of the components are not implemented.

OTM (Object Transaction Manager) layer is not completed yet. It will contain the common features of JDO and ODMG APIs.

OJB provides multiple APIs to the application: PersistenceBroker API, JDO API (as a plugin), and ODMG API. OJB PersistenceBroker API is a minimal API for transparent persistence. It supports O/R mapping, storing of objects to RDBMS, deleting of objects from RDBMS, and retrieval of objects with queries. OJB ODMG API is an ODMG 3.0 implementation; it is built on top of the PersistenceBroker API. This OJB implementation is a full implementation of ODMG 3.0. Transactions are supported; it provided locking for concurrent threads with isolation levels. The OJB JDO API is provided by a plugin which is an implementation by ObjectStore.

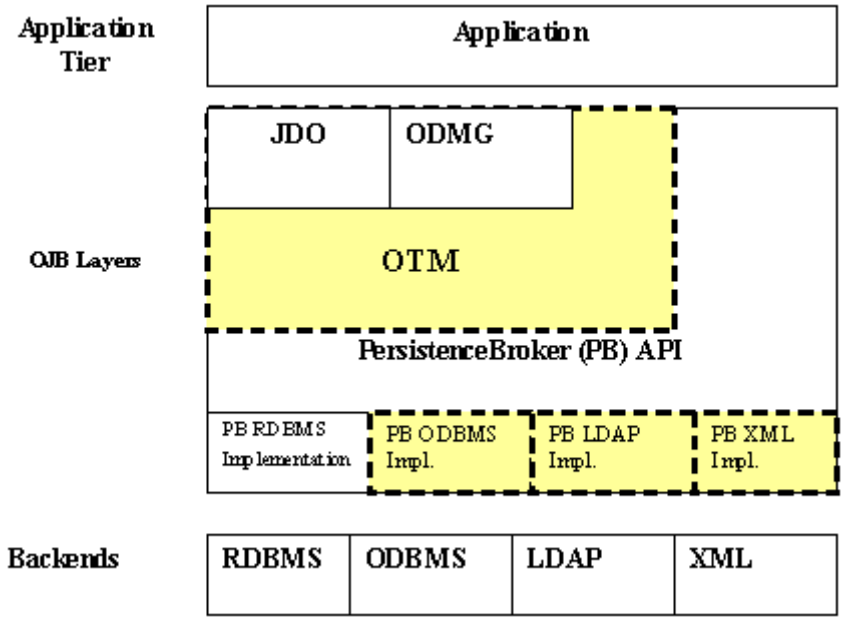

Figure 3-26 OJB architecture

The not-yet implemented layers from the figure are OTM layer, PB ODBMS implementation, PB LDAP implementation and PB XML implementation. These layers are surrounded with broken lines in the figure. They are future to-dos of OJB.

An example mapping for OJB will not be given.

3.5.2.3 Castor JDO

Castor JDO [CJDO] is another O/R mapping framework. It is also an open source project. Although its name suggests support of the JDO standard, it is not a JDO implementation. Castor JDO has also its own query language.

Architecture

The Castor persistence engine handles object persistence, object caching, transaction concurrency and locking. It interacts with the database; it is the interface between the application and the database. The Persistence Engine uses mapping files and database properties set in configurations to interact with database.

There exists only one PersistenceEngine instance, a singleton. Persistence engine for an application may have more than one JDO instance for database interaction. The Java objects can be persisted with any of the instances in the application.

The persistency of objects is cached. This brings more efficient persistency. In Castor JDO, to persist the objects, the users do not implement any kind of interface, but if they want to be notified of the persistency events to implement user-specific actions on save, delete, update etc., there is an interface Persistent which provides the necessary methods to implement in case of persistency events. The methods are to be implemented when the application programmer needs necessary operations when persistency events occur.

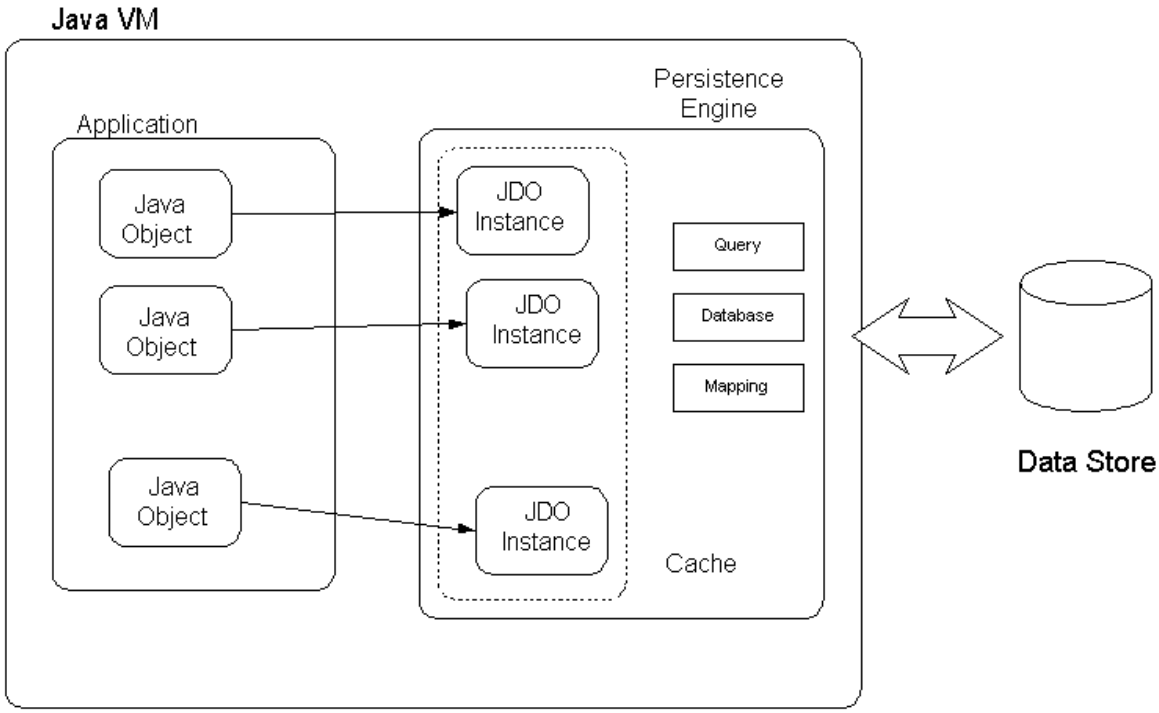

Figure 3-27 Castor JDO Architecture [CJDOa1]

The mapping element seen in Figure 3-27 is representing the mappings for the Java classes to be persisted into database. There are XML documents to define class mappings, also another XML document to specify the database properties.

Castor JDO has its own query language, Castor JDO OQL [CJDO03] which is not JDO compliant either.

3.5.3 Comparison of O/R Mapping Frameworks

In the next subsections, there will be a comparison of the inspected frameworks for O/R persistence. The comparisons help to decide which framework to use for the object persistence in this work.

3.5.3.1 Technical Aspects

In this section, relationally completeness of the frameworks will be analysed. This comparison table is based on comparison from [ORTC03].

	Hibernate	OJB	Castor JDO
Persists arbitrary classes (no special superclass or interface required)	Yes	Yes	Yes
Requires manual SQL building	No	No	No
RDBMS support/independence	JDBC, special for Oracle, DB2, Sybase, MS SQL Server, PostgreSQL, MySQL, HypersonicSQL, Mckoi SQL, SAP DB, Interbase, Progress, Pointbase	JDBC, special dialect support for Db2, Hsqldb, Informix, MsAccess?, MsSQLServer, MySQL, Oracle, PostgreSQL, Sybase, Sapdb.	DB2, HypersonicSQL, Informix, InstantDB, Interbase, MySQL, Oracle, PostgreSQL, SAP DB, SQL Server, Sybase, Generic (for generic JDBC support)
EJB support	Full support for stateful/stateless session beans and BMP entity beans, JTA, JNDI, JMX integration	Full support for SessionBeans and BMP EntityBeans, JNDI, JTA & JCA integration	No
Supports relationships between objects	Yes	Yes	Yes
Mapping supports grouping (GROUP BY clause)	Yes	Yes	No
Mapping supports aggregate functions (count(), avg(), etc.)	Yes	Yes	No
Includes full support of lazy resolution of all queries	Yes, object "faults" (runtime CGLIB bytecode-generation for proxies), lazy relationship reading	Yes + dynamic proxies for all relationships	Yes + dynamic proxies for all relationships
Maintains single identities for objects returned from queries (aka "uniquing")	yes (in the session scope)	Yes	Yes
Generates Mapping as well as the Java Objects themselves, so one doesn't	Yes (support for multiple development models; Java first, mapping first, table	Yes	No

duplicate information in the Java Objects and the related mapping information.	first, Java/XDoclet first) Jakarta ObjectRelationalBridge: Yes		
Supports Composite Primary Keys	Yes (separate primary key class, or "embedded" key as subset of object properties)	Yes	Yes
Aggregate Mappings - Single field maps to multiple fields in database.	Yes	Yes	Yes(limited)
Supports both many to many and one to many associations	Yes	Yes	Yes
Supports collections of Strings, Integers, Dates, etc	Yes (as an actual table association, NOT as a serialized blob)	Yes	Yes
Supports inheritance / polymorphic queries	Yes (all three mapping strategies: table-per-hierarchy, table-per-concrete-class, table-p per-subclass)	Yes	No
Supports one to one associations	Yes (on primary key or arbitrary foreign key)	Yes	Yes
Can fetch associated objects using SQL outer joins	Yes (ANSI and Oracle style outerjoins)	Yes	Yes
Support for optimistic locking / versioning	Yes (version numbers or timestamps)	Yes (version numbers or timestamps)	Yes (timestamps)
Unit of work transactions support	Yes	Yes	Yes
Providing a SUN JDO compliant API	No	Yes(with plugin)	No
Providing an ODMG compliant API	Yes (work in progress)	Yes(not complete OQL)	Yes
Requires "extra" database tables holding locks, metadata, etc.	No (Only one of the key generation strategies requires it)	No (only for a certain sequence number strategy and other special features)	No
Supports multiservers (clustering) and simultaneous access by other applications without loss of transaction integrity	Yes (including clustered cache)	Yes	Not documented
Requires code generation / bytecode processing	Yes, but at runtime, not buildtime	No for kernel and ODMG API, Yes for JDO Api	No
Requires RuntimeReflection	No. (but for object instantiation)	No (user can choose different property access strategies)	Yes
Query Caching - Built-in support (developer writes no code).	Yes. Hibernate Dual Layer Cache Architecture - session level + JVM level caching + query result set caching (also	Yes (PreparedStatement caching and persistent instance caching)	Yes (Persistent instance caching: count-limited | time-limited | unlimited)

	PreparedStatement caching)		
Supports sequences and identity/autoincrement columns for primary key generation (Not using the incorrect SELECT MAX method!)	Yes. both sequences and identity columns (also hi/lo, uuid, etc. as well as user defined strategies)	Yes. (also has hi/lo, uuid, and user defined strategies)	Yes. (also has hi/lo, uuid, sequence, etc)
Supports ternary associations	Yes (and quaternary, etc).	Yes	Not documented
Supports mapping of one class to multiple tables	No	Yes	Yes(via extends relationship)
Supports mapping of multiple classes to one table	Yes	Not documented	No
Supports persistence of properties through private fields	Yes	Yes	No
Supports persistence of properties through accessors (get/set methods)	Yes	Not documented	Yes

3.5.3.2 Other criteria used for comparison

Support

Hibernate has a very large user base, and very active email lists, newsletters, news groups. It is already accepted by a big user community. This can't be said for Castor JDO. Although it is not a new product, it is not very much supported and used, so it is not yet a mature product. OJB is out there for relatively long time, still it has not as many users as Hibernate.

Stability

Hibernate is a stable product, used by many users. Castor JDO is not yet complete, even the documentation of it, this makes the product distant to users. OJB has incomplete documentation. And although the architecture of OJB has many useful components, they are not implemented yet. OJB is still a buggy O/R mapping tool.

Ease of Development

This is similar for all the frameworks inspected. They all have XML configuration files for class-to-database schema mapping. The extra code for object persistence in the Java application is not complex for any of the frameworks. The fact is that all those frameworks are targeted at persisting existing Java objects, so one should better think and create the database schemas according to the java classes. If one has already defined schema which one should not change and one already designed the Java classes, then using the frameworks are kind of difficult. The reason is that the frameworks are not flexible enough for many features when mapping.

In case there is the need to persist the objects of an application, the best is to create the schemas the way the frameworks suggest. This makes mapping very simple. Hibernate for

example has a schema generator component which can generate database schemas from the defined mappings.

Other than this, if the tables do not have key columns, using all those frameworks is either not possible or they do not persist and query objects properly.

Performance

Performance of Hibernate is at most 10% less than the simple JDBC calls; Hibernate performs caching and optimizations for database connections and queries, which would be very difficult to program for developers.

Castor JDO has also a cache, it performs caching for persistency actions, but still it is not very effective. OJB is also not as good as Hibernate in performance.

3.5.3.3 Selection of Framework

After inspecting the frameworks and comparing them, I decided to use Hibernate. The main reasons are that it has very high performance and has a very large user base, already accepted by a huge group of users. Hibernate also has very good, detailed documentation. In case one is stuck in mapping of classes or any other problem, there is a very active user mailing list for finding the answers to problems. From my view, the only barely seen disadvantage of Hibernate is that it has its own API both for persistence and querying of the objects. It is not JDO compliant and it is not ODMG compliant either (although it is in progress). This can be a big problem for large software projects if one day the persistence is changed from Hibernate to any other framework. Choosing a JDO compliant tool can be great help.

But one other question is whether JDO or OMDG compliancy is a big advantage and being implementations of JDO or ODMG is a big disadvantage. JDO still is an immature, flawed standard, and there are not many free licence implementations of it, and the ones which are free are generally incomplete. ODMG standard is also a similar standard. Although it tries to solve the persistency of objects transparently, it can not be called successful. Both JDO and ODMG standard are kind of unused standards. None of the OO relational mapping vendors were members of the specification committee. The committee was mostly composed of ODBMS vendors. This is maybe the main reason that the standards are not used very much. The requirements were not done by the real “customers”. JDO is not a standard for Java persistence; rather it’s a standard API for ODBMS [object oriented database management system]. The standard itself is not supported by any of the major relational database vendors (i.e. IBM, Oracle, Sybase, etc.).

4. System Design

In this chapter the overall architecture of the system will be explained. Short explanations of the components will follow with the roles they have in the system.

4.1 Architecture

The system is composed of many frameworks and components: Tomcat Servlet Container, Struts framework, OXF framework, Hibernate, Castor XML mapping, MySQL database, Commons DBCP, Commons Pool and web client browser.

The main components of the system are the client browser, servlet container controlling the life cycle of web application and the database.

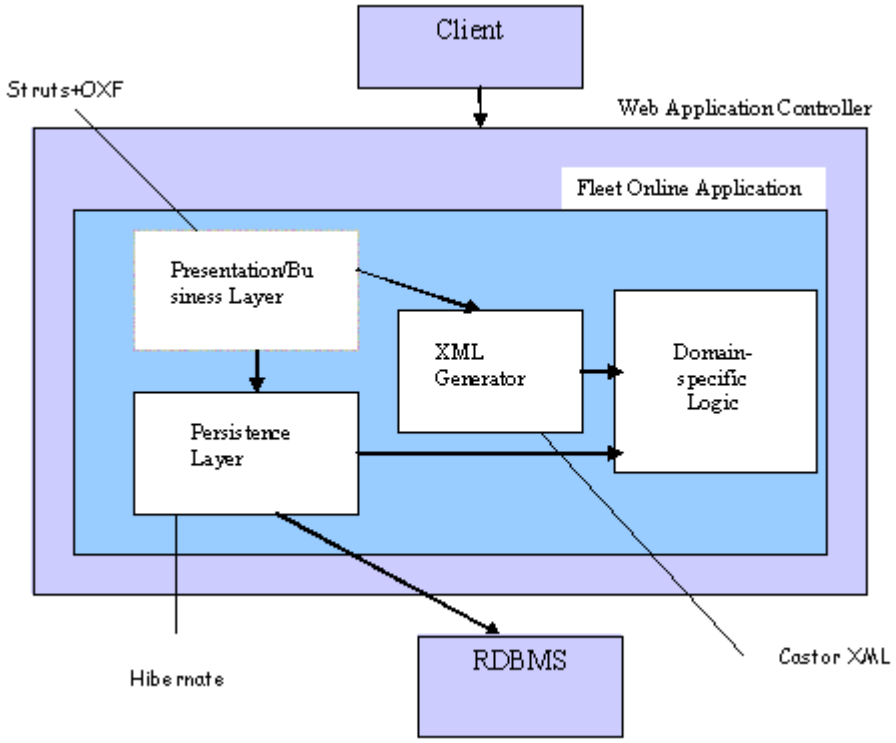

Figure 4-1 Architecture of the System

The client is any Web browser to access the Fleet Online application. The servlet container Tomcat is a container to run the web applications according to Sun's specifications. The web application GL fleet online runs as a component in control of Tomcat.
The web application component is composed of many packages, mapping files, static resources.

In the next subsections the specific components in the architecture will be shortly explained.

4.2 The Components

In the previous section, the overall software architecture of the system is explained. In this section, the components will be explained more detailed excluding the implementation details. The tasks and responsibilities of the components in the overall architecture and how they work together will be made clear.

4.2.1 Client

Client browser is any browser client(user) used to send HTTP requests to login to fleet online, to log off, to see the status of the fleet belonging to the specific user, and finally to see the current information about a specific survey.

4.2.2 Web application Container

Tomcat [TmCT03] is the Web application Container which implements the Sun's specifications about servlet containers. The Tomcat server is a Java-based Web Application container that was created to run Servlet and JavaServer Pages web applications. It has become the reference implementation for both the Servlet and JSP specifications. It provides many facilities for the web applications other than main life cycle actions. The application fleet online is a deployed web application running under the Tomcat container. The version of the Tomcat used is 4.1.24. Tomcat also provides facilities like pooling for database connections, static resource automatic loading, etc.

4.2.3 Persistence Manager

Hibernate is used for O/R mapping. The tables Fleet Online application related tables in the database will be mapped to corresponding classes from the domain model of the application. The database model and the relationships will be shown in the implementation chapter.

4.2.4 Presentation/Business Logic

The presentation and business logic are implemented by components Struts and OXF.

Struts component comprises Struts framework classes and classes implemented/extended by Struts framework classes.

The business logic actions and the request dispatching for the web application are processed by this component. This package is composed of many classes implementing Struts functionality. Struts component has a dependency to Hibernate component in the architecture. For implementation of business logic business data which resides in RDBMS is used. The task of database interactions and object persistence functionality is provided by Hibernate. Hibernate itself uses XML configuration files to map the tables to specific classes of the application. The details are explained in implementation chapter.

In this architecture, OXF framework is used for HTML views created by XSLT and XML. When it is necessary, Struts forwards requests to OXF pipeline which is handled by OXF Processor servlet. OXF pipelines are used in this architecture just for view creation. In this web application architecture the data transfer between Struts framework and OXF framework is done by saved objects in HTTPSession. In the OXF pipeline, a processor gets the specified bean from the current session object and creates the XML document by using Castor XML mapping framework. The next step in the pipeline is calling the XSLT processor and creating the views which are HTML pages.

4.2.5 RDBMS

MySQL is used for persistence. Hibernate framework provides classes and interfaces to access database related resources.

Connections to MySQL are pooled and as said previously. Tomcat Servlet Container uses Commons DBCP to pool the connections and then through the interfaces provided the application can have reference to pooled connections.

4.2.6 XML Generator

Castor XML is one of the components in the architecture. Castor XML is used for mapping of Java objects to XML documents. After XML structure creation with Castor XML, XSL transformations are applied to the result XML documents to produce the corresponding HTML views.

In the architecture, OXF component uses the Castor XML. In the OXF pipeline, a processor gets the Java beans saved under the session for the customer and uses Castor XML to create the XML documents which will be the input given to the next processor in the pipeline which is an XSLT processor.

4.2.7 Domain Model Classes

The Domain Model classes exist under the package `org.glgroup.survey.model`. Those classes are the heart of the application. They hold the main business data and logic. In this architecture, because the Persistent classes are persisted by Hibernate, not by programmer via JDBC calls, some of the Business Data classes are changed. For most cases, Hibernate can map existing classes to existing tables, but there are special cases where either the class definition or the table definition lacks necessary properties or columns. In this case, the client programmer has to change either the database tables or Java Classes. Because the database tables of Germanischer Lloyd are already being used by existing applications, it was least desired to change the tables even when they are not good modelled. To enable OO representation of data residing in database, some Domain Model [Fowler03] classes are needed to implement specific interfaces of Hibernate framework. The implementation details will be explained later.

4.3 Sequence Diagrams

The interaction between the various components of the system can be seen in the following scenarios. In Figure 4-2 the login of the user to the system is shown, and in Figure 4-3 viewing the survey request for a specific ship. Here the main components' interactions are shown for login to fleet online and for viewing a selected survey request.

In UML sequence diagram shown in Figure 4-2, the overall interaction between the client, WAC (Web Application Controller), the web application and RDBMS for login request can be seen. A client initiates a logon request for fleet online with the fleet online URI, and then the Tomcat Servlet Container receives the request and executes the corresponding web application for the request, which is fleet online web application. Then the web application connects to RDBMS and gets the customer object with the given username and password. The username and password are sent by client with HTTP request. Then from the RDBMS the details of the logged in customer are received. The ships and survey requests belonging to the ships are also retrieved from RDBMS. After this, the web application creates the HTML page showing the fleet status page for the customer. This page shows the ships and overall details for the survey requests of the ships. If the customer login action is successful, the session object for the customer is created which contains the customer data and fleet data for this customer.

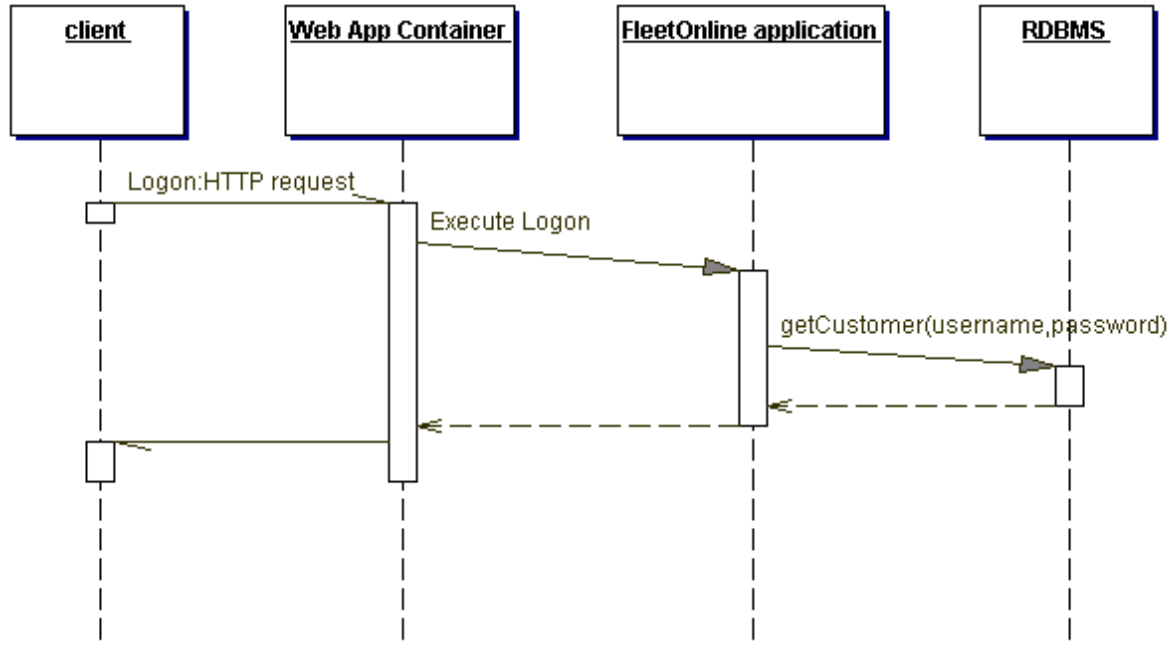

Figure 4-2 Sequence Diagram-Logon to Fleet Online

The next main interaction between client and the web application is viewing the details for a selected survey request of a selected ship which is seen in Figure 4-3. From the fleet status page, the client (customer) can select a specific survey request to view the details. The client selects the survey request and sends the HTTP request. WAC gets the request and selects fleet online web application to execute. Fleet online application has already a created session

object for the customer which contains the ships and surveys for this client. It does not need to get the data from the RDBMS again. The HTML page for viewing the details for selected survey request is created and it is sent to the customer (client) as HTTP response.

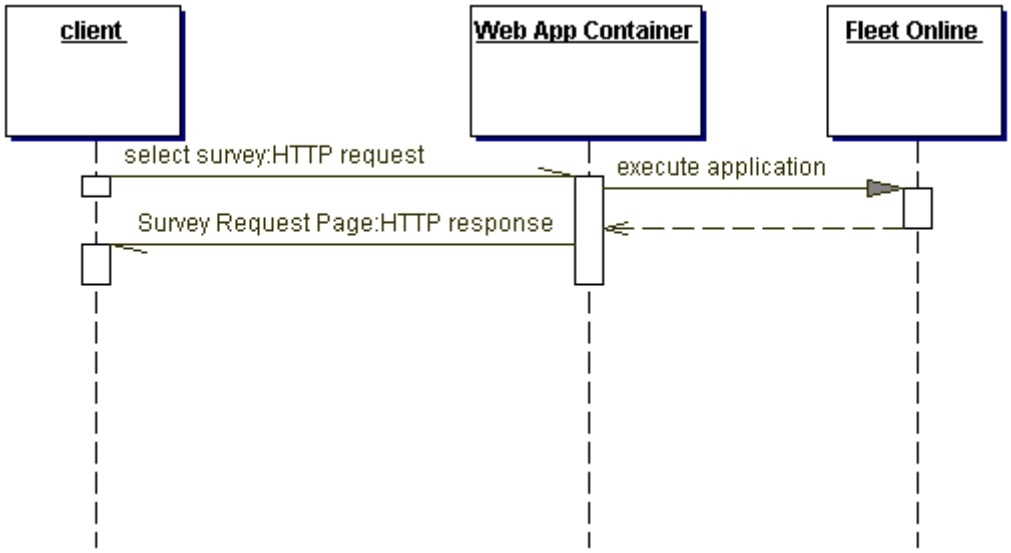

Figure 4-3 Sequence Diagram-Show Survey Request

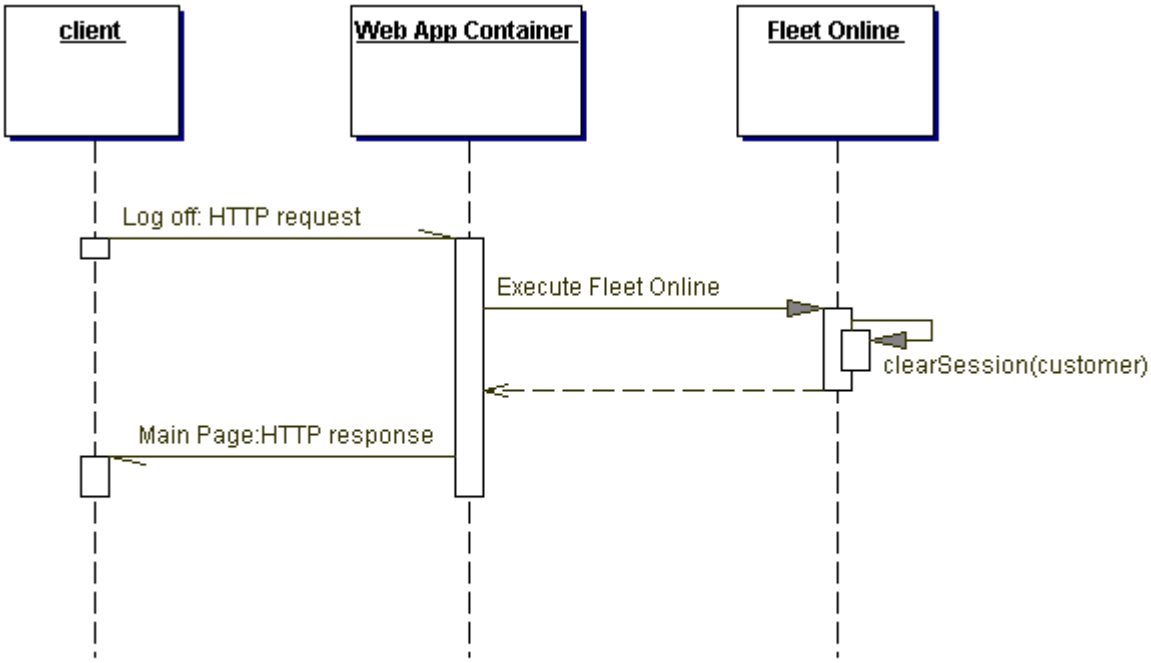

Figure 4-4 Sequence Diagram-Logoff from Fleet Online

The last interaction between client and the web application is the logoff action. The client wants to logoff from the web site. He sends a logoff request. WAC first gets the request, executes the web application. The application clears the session object for the customer and sends the HTTP response to the client. The response page sent is the home page of the fleet online application. The UML sequence diagram of this interaction can be seen in Figure 4-4.

5. Implementation

In this chapter the implementation details will be presented. The class diagrams, use of Castor XML, Hibernate, OXF, Struts, etc. will be shown.

5.1 Web Application in Java

With the release of the Java Servlet Specification 2.2, the concept of a web application was introduced. According to this specification, a "Web Application is a collection of servlets, html pages, classes, and other resources that can be bundled and run on multiple containers from multiple vendors." [Servlet03]

At the heart of all web applications is a *deployment descriptor*. The deployment descriptor is an XML file named *web.xml*. It describes configuration information for the entire web application. The information that is contained in the deployment descriptor includes the following elements:

- ServletContext Init Parameters
- Localized Content
- Session Configuration
- Servlet / JSP Definitions
- Servlet / JSP Mappings
- Mime Type Mappings
- Welcome File list
- Error Pages
- Security(Authentication)

The application GL Fleet Online service is a web application running in the Tomcat Servlet Container. As a Java Web application, it comprises many components and elements. The central connection of all is the deployment descriptor of the Fleet Online web application.

In sections 5.2 and 5.3 the configurations made for deploying the Fleet Online web application under Tomcat Servlet Container is explained. The Fleet Online web application uses Commons DBCP [DBCP03] and Commons Pool [CoPool03] components database connections pooling. To enable Fleet Online to access these resources, they should be registered in Tomcat's server.xml configuration file. In section 5.2 details are explained.

In section 5.3 there will be explanation about the deployment descriptor for Fleet Online web application. Servlets, JSPs and many elements existing under the web application are specified in this deployment descriptor. Without the deployment descriptor, Tomcat would not be able to properly serve the Fleet Online web application deployed under it.

In section 5.4 the Struts configuration file (struts-config.xml) details will be explained. This configuration file is the main component of a Struts-based web application. Through it data sources, form beans, action mappings and global forwards are defined.
Only the selected details will be presented.

In section 5.4 implementation of the main Struts functionality will be explained. The Struts Actions, use of OXF pipelines, domain model classes used for Fleet Online and Castor XML mappings are the main points where implementation will be explained in more detail
.
In section 5.X the Hibernate mappings will be explained. All of the O/R mappings done for domain model classes will be given in this section. At application start-up the mapping files are read and the O/R mapping layer is made available to the application program. Hibernate classes are used/instantiated in Struts `Action` classes in our architecture.

Finally in this section a detailed sequence diagram will be given in section 5.6. Section 5.7 is the last section of this chapter where snapshots of the web interface are shown and explained briefly. Here JSP views and OXF pipeline output views which are created by applying XSL transformations to XML documents can be seen clearly.

5.2 Deploying the Web Application under Tomcat

The component Commons DBCP should be included in the application to enable efficient database interactions.

To enable Tomcat use the Commons DBCP component, the following configurations seen in Table 11 should be done in server.xml configuration file of Tomcat. The elements <Resource> and <ResourceParams> should be added and configured to context of the *Fleet Online* web application. The configurations done in server.xml of Tomcat for Fleet Online web application can be seen in the Table 11.

```
<Context path="/fleetonline"  docBase="C:\allWebApplicationDevelopment\jakarta-
tomcat\jakarta-tomcat-4.1.24\webapps\fleetonline"
workDir="C:\allWebApplicationDevelopment\jakarta-tomcat\jakarta-tomcat-
4.1.24\webapps\fleetonline\work\org\apache\jsp"  reloadable="true"
crossContext="true">
 <Resource name="jdbc/fleetonline" auth="Container" scope="Shareable"
type="javax.sql.DataSource"/>
 <ResourceParams name="jdbc/fleetonline">
        <parameter>
            <name>factory</name>
            <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </parameter>
        <!-- DBCP database connection settings -->
        <parameter>
            <name>url</name>
            <value>jdbc:mysql://localhost/gl</value>
        </parameter>
        <parameter>
            <name>driverClassName</name>
            <value>com.mysql.jdbc.Driver</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>root</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>pass</value>
        </parameter>
        <!-- DBCP connection pooling options -->
        <parameter>
            <name>maxWait</name>
```

```
            <value>5000</value>
        </parameter>
        <parameter>
            <name>maxIdle</name>
            <value>2</value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>4</value>
        </parameter>
    </ResourceParams>
  </Context>
```

Table 11 Resource Integration in Fleet Online application

5.3 Deployment descriptor of Fleet Online web application

One most important deployment configuration is stating the servlet properties, names, initial parameters to set and mappings. In this web application, there are two servlets : *Struts* and *Oxf* servlets. Deployment descriptor file can be seen in Appendix.

The Struts servlet serves URIs with those properties: Application root URI or URIs ending with *.do. Oxf servlet serves static resources like images, CSS, etc., and URIs ending with ".xpl". The XML pipelines reside in files ending with ".xpl" as convention.

There are few other configuration properties configured in deployment descriptor for the web application. Context for the application, data source properties for the application are examples for them.

5.4 Struts Configuration file

Struts configuration file is used to specify Struts-specific properties and action mappings. Action mappings are used by Struts controller servlet which is an instance of `org.apache.struts.action.ActionServlet`. Its purpose is to control the web application. In our architecture, not only Struts but also OXF serves some of the URIs. Still, the controller task is done by Struts. The Struts controller forwards the requests to OXF servlet whenever there are requests for pipelines for view creation.

Here a snippet of the Struts configuration file is given. The controller is mainly configured by this snippet. For example if a request is sent for main page of *fleetonline*, the request is forwarded to *index.jsp.* When there is a request to "/logon", the request is forwarded to an instance of `org.glgroup.fleetonline.struts.LogonAction`. For the details of the configuration file, please refer to the Appendix.

```
<action-mappings>
        <action path="/" forward="/index.jsp"/>
      <action    path="/logon"
                type="org.glgroup.fleetonline.struts.LogonAction"
                name="logonOXFForm"
               scope="request"
               input="logon.jsp">
      <forward name="success" path="/logon.jsp"/>
      </action>

       <!-- Process a user logoff -->
```

```
    <action    path="/logoff"
               type="org.glgroup.fleetonline.struts.LogoffAction">
             <forward name="success" path="/index.jsp"/>
    </action>

    <action    path="/showfleet"
               type="org.glgroup.fleetonline.struts.ShowFleetAction">
             <forward name="success" path="/ShowFleet.xpl"/>
    </action>
    <action    path="/showsurveyrequest"
type="org.glgroup.fleetonline.struts.ShowSurveyRequestAction">
             <forward name="success" path="/ShowSurveyRequest.xpl"/>
    </action>
    </action-mappings>
```

Table 12 A snippet of struts-config.xml showing the action mappings

5.5 Struts functionality

5.5.1 Domain Model Classes and Database Tables

The business logic and business data resides in the Domain Model classes. In GL Fleet Online Service business logic which is implemented in the student project is not complex. As analyzed in the chapter 2, login, logoff, view fleet status and view survey request are the main business logic actions to implement. The class diagram for business objects is in Figure 5-1.

The customer information resides in the class `Customer`. The attributes like username, password, name, address, id etc. can be accessed by `Customer`. The `ContactInfo` wrapper class hold the contact information of the customer.

The information for a ship is represented by the class `ShipData`. The register number, flag, etc., properties can be found here. The class `CustomerShip` holds the link between the customer and his ships. An instance of `CustomerShip` class holds the customer id and one of the ship's ids he has. There is a 1: n relationship between `Customer` and `ShipData`. `Customer` class holds many instances of `CustomerShip`.

The `SurveyItem` class holds information of survey items. The class `OrderedSurveyItem` holds a reference to `SurveyItem` instance and some specific information about the ordering of the customer as due date, event on, and the state of the order. There is a 1:1 relationship between an `OrderedSurveyItem` and `SurveyItem`.

A survey has a list of survey items. The class `OrderedSurvey` represents a survey which is ordered by the customer. The 1:n relationship between the survey and its items can be seen in the class diagram between `OrderedSurvey` and `OrderedSurveyItem` class.

A customer may have many surveys ordered for specific ships of him. So, there is a 1:n relationship between the customer and his surveys. `Customer` and `OrderedSurvey` class has an 1:n relationship.

An `OrderedSurvey` will be evaluated by a surveyor. The class `Person` represents the main information for a person as name, surname and contact information. An `OrderedSurvey` has an instance of `Person` which represents the surveyor responsible for the survey. So, `OrderedSurvey` has a dependency to the `Person` class.

An owner of a ship is represented by the class `Owner`. This class has information as owner company details, contact information details and contact person details. A ship has an owner and there is composition between these two objects.

The class `DateTime` is a helper class for O/R mapping of a Java date field which is represented in the database in a special string format. This brings the need of writing a special handler for mapping between the Java date object and the database column. As the O/R mapping framework Hibernate was chosen, a special Java interface class defined for handlers is implemented, i.e., the interface `net.sf.hibernate.UserType`. The class `DateTimeType` implements this interface and the class `DateTime` extends the `DateTimeType`. `DateTime` type is a JavaBean-similar class with getters and setters. The methods to map the objects of type `DateTime` to database comes from the implemented class `DateTimeType`.

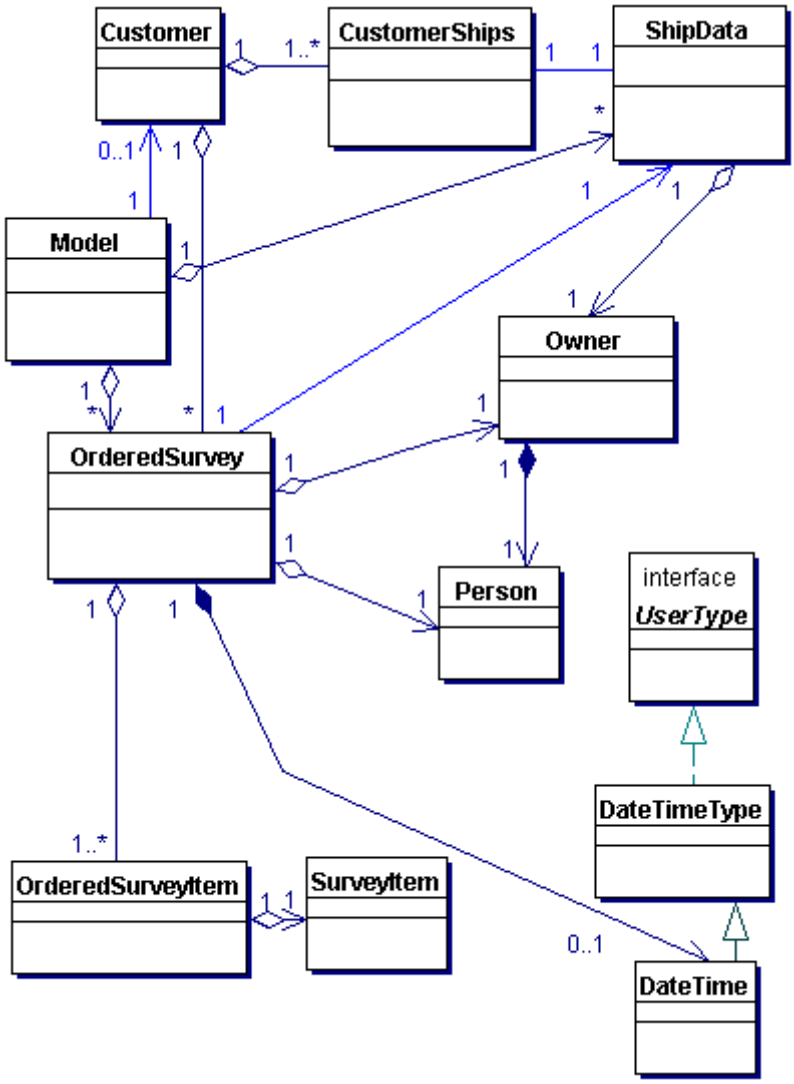

Figure 5-1 org.glgroup.survey.model package class diagram

The business objects of Fleet Online are persisted to database. The database tables representing the Domain Model classes can be seen in Figure 5-2.

The table glis_customer represents the customer information. The dbnr column represents the unique id for the customer. This column also provides the link between a customer and the ships he has. The glis_customer table has customer name information, user account information, number of ships he has etc.

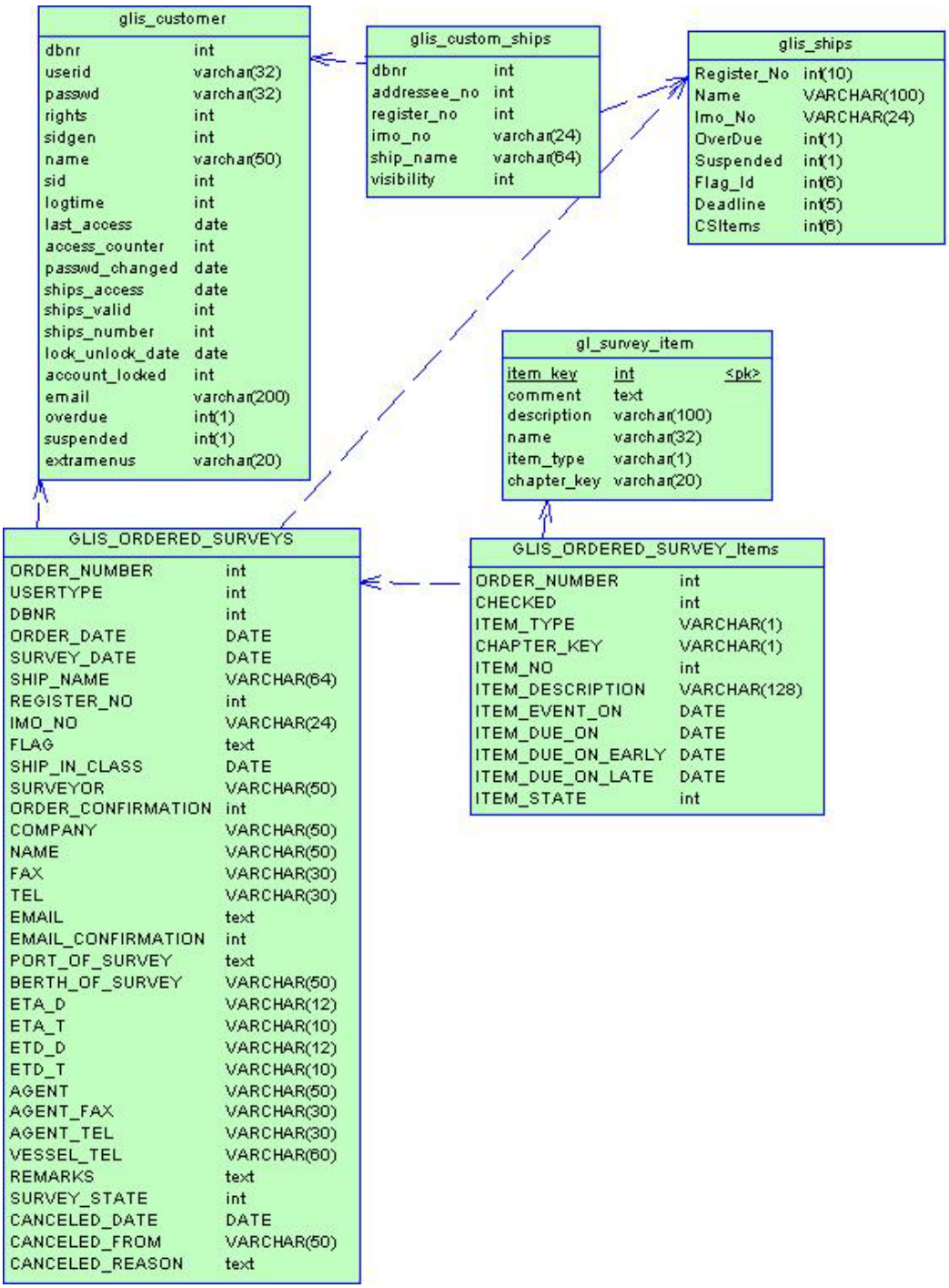

Figure 5-2 Database schema with the relevant tables

The glis_ships table represents a ship's information. The register_no column of a ship uniquely defines this ship. The rest of the columns in this table represent additional information about the ship.

The table glis_custom_ships provides the linkage information between a ship and the customer having the ship in his fleet. The columns dbnr and register_no provide this linkage.

The column dbnr of this table, i.e. glis_ships table, references the dbnr column of glis_customer table. Similarly the column register_no references the register_no column of glis_ships table. The rest of the columns are representing additional information about this linkage. This table has a composite key (dbnr, register_no). The dbnr referencing the customer can exist in more than one row as a customer may have more than one ship.

The table gl_survey_item maps to SurveyItem Java class. The column item_key uniquely defines a survey item. As mentioned before, an ordered survey item has one survey item and additional information about the ordering action like due date of the order etc. The table glis_ordered_survey_item represents an OrderedSurveyItem object. The item_no column in this table references the column item_key in the table glis_survey_item.

The glis_ordered_surveys table represents an instance of OrderedSurvey. The column order_number in this table is the key. An ordered survey may have many ordered survey items. The column order_number in glis_ordered_survey_items table references the order_number column of glis_ordered_surveys. As stated before a customer may have ordered many surveys. This information is represented by foreign keys in the database schema. The column dbnr of glis_ordered_surveys table references the dbnr column of glis_customer.

Classes in org.glgroup.survey.model Package	**Database Tables**
`Customer`	glis_customer
`SurveyItem`	gl_survey_item
`OrderedSurveyItem`	glis_ordered_survey_items
`OrderedSurvey`	glis_ordered_surveys
`ShipData`	glis_ships
`CustomerShip`	glis_custom_ships

Table 13 Class-to-tables mappings

In Table 13 the mappings of the classes to the database tables are summarized. In the database schema, there are foreign key constraints representing the relations between the tables as described above. They can be seen in Table 14.

Column	**Table**	**Referenced column**	**Referenced table**
dbnr	glis_custom_ships	Dbnr	glis_customer
register_no	glis_custom_ships	register_no	glis_ships
dbnr	glis_ordered_surveys	Dbnr	glis_customer
order_number	glis_ordered_survey_items	order_number	glis_ordered_surveys
Item_no	glis_ordered_survey_items	item_key	gl_survey_item
Register_no	glis_ordered_surveys	register_no	glis_ships

Table 14 Foreign key relations of the fleet online database schema

The tables are designed by [FTEAM03]. In some cases there was the necessity to change the tables' definition especially relating the primary keys and foreign keys. In these cases the

tables are slightly changed with the consideration of minimum changes. The most important change is seen at glis_ordered_survey_items table. Before the changes this table did not have a key, many rows having the same data could be existing. This is unsuitable for Hibernate mapping. In Hibernate every business object to be mapped needs a unique id. So, key column id is added to this table.

5.5.2 JSPs

In this project there are only two main JSP views. As stated before, remaining views are created by applying XSL transformations to XML documents.

The first JSP view is index.jsp which creates the main entrance page to GL Fleet Online service. Currently this page is quite simple; there is only a link to login page. The second JSP view is the logon page. On this login page there is an HTML form which gets the username and password from the user.

5.5.3 Implemented Struts Actions

An `Action` is an adapter between the contents of an incoming HTTP request and the corresponding business logic that should be executed to process this request. The controller will select an appropriate `Action` for each request, create an instance (if necessary), and call perform method of `Action`.

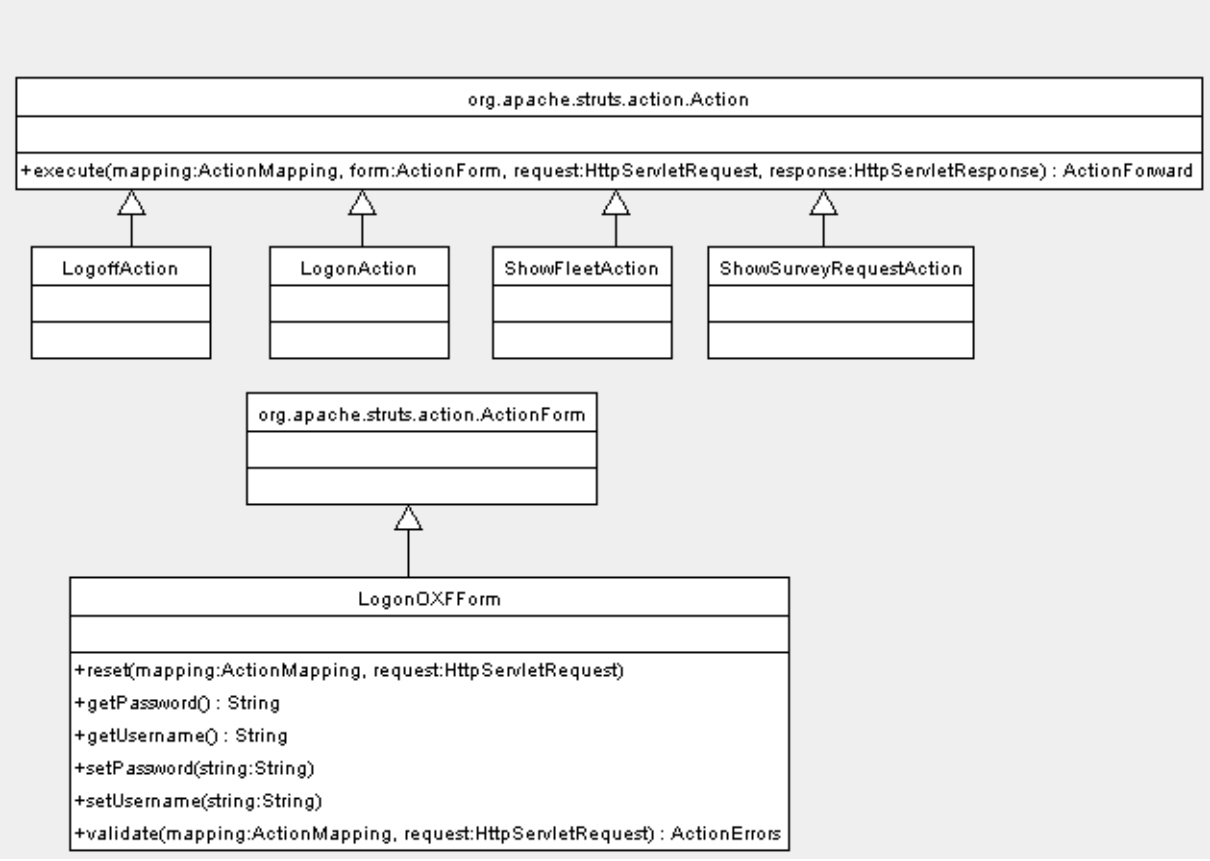

Figure 5-3 Actions and ActionForm classes

In Fleet Online web application, there are four main requests from the user, they are login to fleet online, logoff from fleet online, viewing the general fleet status after the user's logon and viewing the selected survey request details for a selected ship of the user. For these main requests, there are 4 action classes implemented. `LogonAction, LogoffAction, ShowFleetAction` and `ShowSurveyRequestAction` classes in package `org.glgroup.fleetonline.struts` do the specified actions. The class diagram for this package can be seen in Figure 5-3.

In this package the class LogonOXFForm is a class which extends the ActionForm. ActionForm is a class from Struts library to ease the HTML form input handling. LogonOXFForm class eases the task of retrieving the username and password from a login form.

5.5.4 OXF Pipelines

There are two OXF pipelines for output creation. The first pipeline creates the fleet status page and the second the survey request page for a logged-in user whose session already contains data about the ships in his fleet and the corresponding survey requests.

5.5.4.1 ShowFleetStatus pipeline

This pipeline creates the fleet status page for a logged-in customer. There are two processors in the pipeline, a bean-generator processor and an XSLT processor. The bean-generator processor has two inputs and one output. The first input is an instance of org.glgroup.survey.model.Model for the customer which resides in the session. The second is the Castor XML mapping file. The output is the XML representation of the model instance. This XML representation is created by using Castor XML mapping files.

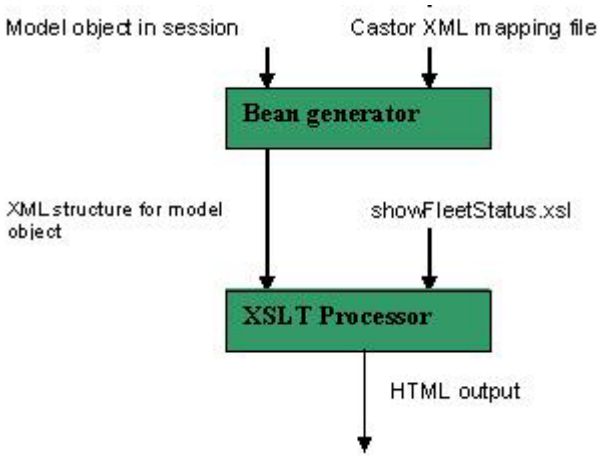

Figure 5-4 ShowFleetStatus pipeline

The second and last processor in the pipeline is XSLT processor. XSLT processor has two inputs and one output. The first input is the XML structure to which XSL transformation will be applied, and the second input is the XSL stylesheet to apply. The first input comes from the previous processor's output, i.e., from bean-generator processor which is the XML structure

for model object. After applying the transformation, the output page which will be sent to the customer is created.

Table 15 shows the pipeline expressed in XPL (XML Pipeline definition language).

```
<p:config xmlns:p="http://www.orbeon.com/oxf/pipeline"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <p:param type="input" name="model"/>
  <p:param type="output" name="data"/>

<!--  bean-generator processor definition -->
<p:processor uri="oxf/processor/bean-generator">

<!-- the first input, the bean object in the session for customer-->
 <p:input name="config" xmlns:p="http://www.orbeon.com/oxf/pipeline">
   <config>
     <attribute>model</attribute>
     <source>session</source>
   </config>
 </p:input>

<!-- the 2nd input, Castor XML mapping file-->
  <p:input name="mapping" href="oxf:/mappingGL.xml"/>

<!-- the output of this processor, XML structure for the model bean in session-->
 <p:output name="data" id="modelSession" debug="modelSession"/>
  </p:processor>

<!-- XSLT processor definition -->
<p:processor uri="oxf/processor/xslt">

<!—the first input which is output of previous processor -->
      <p:input name="data" href="#modelSession"/>
<!—2nd input, xsl stylesheet to apply-->
       <p:input name="config"
        href="oxf:/style\showFleetStatus.xsl">
        </p:input>
<!-- output HTML page created which will be sent to user-->
      <p:output name="data" ref="data" />
  </p:processor>
</p:config>
```

Table 15 ShowFleetStatus pipeline definition in XPL

5.5.4.2 ShowSurveyRequest pipeline

The second pipeline used in the web application creates the survey request page for a logged-in user. It has the same processors as the previous pipeline, i.e., bean-generator processor and XSLT Processor. In this pipeline, only the second input to the XSLT processor is different, it is not the stylesheet to create fleet status page, but the one used to create the survey request page for the customer.

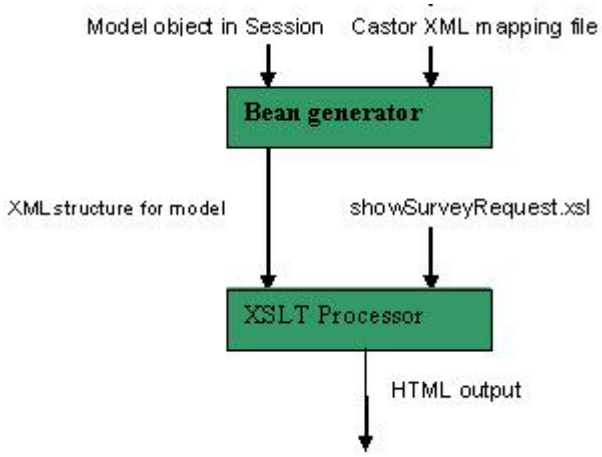

Figure 5-5 ShowSurveyRequest Pipeline

The XPL definition for this pipeline will not be given as it is very similar to the previous definition given in previous section.

5.5.4.3 Castor XML Mappings

Castor XML mappings are used in the OXF pipelines as input for bean-generator processor. The domain model [Fowler03] classes in package `org.glgroup.survey.model` are represented as XML structures. For all the classes in this package an XML representation mapping is created.

The Model class in this package comprises customer and ships and survey requests of this customer. When a customer logs in, an instance of Model having information about the customer and fleet status and surveys information for the customer is created and saved to session for the customer. Mainly the aim is to have an XML representation for this object which is instance of the Model class.

Only an example mapping for the `Model` class will be given, for the mappings of the remaining classes in `org.glgroup.survey.model` package, please refer to the Appendix.

The mapping for class org.glgroup.survey.model.Model is defined by element <class>. It is mapped to XML element <Model>. The field `customer` of this class is mapped to element <Customer>. The mapping for `Customer` class is defined by another <class> element (which is not shown here). The `shipWrappers` field of `Model` class which has objects of type `org.glgroup.survey.model.ShipWrapper` is mapped to <ShipSurveyRequests> element. Table 16 shows the mapping for Model class.

```
<class name="org.glgroup.survey.model.Model" auto-complete="false">
     <map-to xml="Model"/>
     <field name="customer" type="org.glgroup.survey.model.Customer">
            <bind-xml name="Customer" />
        </field>
     <field name="shipWrappers" type="org.glgroup.survey.model.ShipWrapper"
                                container="false" collection="arraylist">
            <bind-xml name="ShipSurveyRequests" />
     </field>
</class>
```

Table 16 Castor XML mapping for class Model

So, the resulting Model class will have such a XML representation:

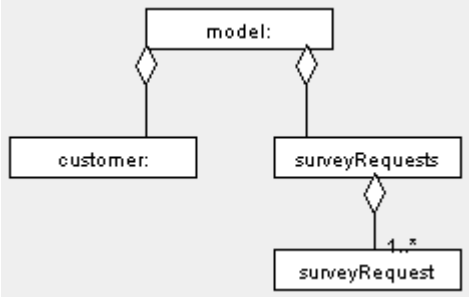

Figure 5-6 Model instance representation

Model has child elements <Customer> and <ShipSurveyRequests>. Both of the child elements have their own children and attributes representing their own structure. <ShipSurveyRequests> element represents a shipWrappers field which is an array list. So, it has same number of children as it is in the list. The children have all same structure where only the values of attributes or the elements differ.

5.5.4.4 XSL Stylesheets

There are two main XSL stylesheets used by OXF pipelines, showFleetStatus.xsl and showSurveyRequest.xsl. Given the XML structure for the model instance for the current customer (which contains ships and survey requests information), showFleetStatus stylesheet renders the HTML output showing the fleet status for this customer. ShowSurveyRequest.xsl renders the HTML page showing the details for a chosen survey of a ship.

5.6 Hibernate Mapping

The purpose of Hibernate is the persistence of application objects, automatic saving/updating to RDBMS and querying and initializing objects from RDBMS.

Hibernate uses XML mapping files for automatic Java objects saving/updating/retrieving to/from RDBMS. In the next sections the mappings for the classes to database tables will be briefly explained.

Please note that in the class diagrams, not-shown cardinalities are *1* and the navigability is not shown with arrows when both directions are navigable.

5.6.1 Class ShipData

This class maps to glis_ships table in RDBMS. The unique ids for the objects of this class are set by programmer. The <id> element specifies the unique id for the objects of this class. It should map to the key attribute of the table. All objects must have unique ids. The `registerNumber` field of `ShipData` represents the unique ids. `name, suspended, csItems, imoNumber, flagId` mappings are straightforward, and can be seen in the mapping document. The mappings of the fields to columns are expressed by <property> elements in the mapping file as they are of basic types.

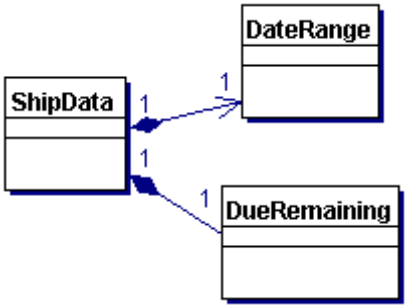

Figure 5-7 ShipData relationships

For mapping of `classPeriod` and `dueRemaining` fields, <component> element is used. The <component> element maps properties of a child object to columns of the table of a parent class. <component> element can map properties which are not instances of simple data types to columns in the table. The `classPeriod` field is of type `DateRange`. The `from` field of `DateRange` maps to ship_in_class column in the table. The `dueRemaining` field has type `DueRemaining`. The `overdue` field of `DueRemaining` class maps to column overdue in the table and the field `deadline` maps to deadline column in the table. The relationships of `ShipData` with `DateRange` and `DueRemaining` classes can be seen clearly in Figure 5-7.

Table 17 shows the mapping definition for `ShipData` class.

```
<hibernate-mapping>

    <class name="org.glgroup.survey.model.ShipData" table="glis_ships">

        <!-- A 32 hex character is our surrogate key. It's automatically
            generated by Hibernate with the UUID pattern. -->
        <id name="registerNumber" type="string" unsaved-value="" >
            <column name="register_no" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
        </id>
        <property name="name" type="string">
                    <column name="name" sql-type="varchar" />
        </property>

        <component name="classPeriod" class="org.glgroup.survey.model.DateRange">
                    <property name="from" type="date">
                           <column name="ship_in_class" sql-type="date"/>
                    </property>
          </component>

        <component name="dueRemaining"
class="org.glgroup.survey.model.DueRemaining">
         <property name="overdue" type="boolean">
                    <column name="overdue" sql-type="int(1)" />
         </property>
        <property name="deadline" type="string">
                    <column name="deadline" sql-type="int" />
             </property>
        </component>

        <property …>
             …
        </property>
    </class>
</hibernate-mapping>
```

Table 17 Hibernate mapping for ShipData class

5.6.2 Class Customer

In this section the mapping of `Customer` class is explained. Not all of the mappings will be shown, but the most important ones. The mappings of selected fields are explained in detail.

`Customer` class maps to table glis_customer which has primary key dbnr column. In the `Customer` class, the field `dbNumber` maps to dbnr column. Fields `userId, password, accessRights, sidgen, sid, name, logtime, lastAccess, accessCounter, passwordChanged, shipsAccess, overdueItems, shipNumber, shipsValid, suspended, lockUnlockDate, accountLocked` map to simple columns in the table.

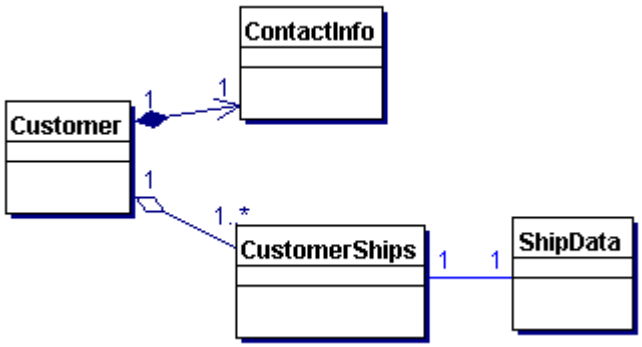

Figure 5-8 Customer Relationships

The field `contact` of `Customer` class is not of a simple type, but of type `ContactInfo`. For the time being, only the email of the customer is saved to RDBMS, so `email` field of `ContactInfo` class maps to email column in the table. Figure 5-8 shows the relationships of `Customer` class with the classes it has relationships.

`Customer` class has a field `customerShips` which is a list. The elements in the list are of type `org.glgroup.survey.model.CustomerShip`. The `CustomerShip` objects reside in the RDBMS in table glis_custom_ships. This table represents the 1: n relationship between the customer and ships with additional properties about the relationships. The column dbnr in glis_custom_ships table references the dbnr column (which is the primary key) in glis_customer. The column register_no in glis_custom_ships references register_no (primary key) in glis_ships. Together with dbnr column, register_no column in glis_custom_ships represents the customer-ships (1: n) relationship. For this a helper class `CustomerShip` is used. The field `ship` of `CustomerShip` class is of type `ShipData`. `ShipData` is also a class persisted to RDBMS, so a new mapping is not necessary, the element <many-to-one> does the mapping of `ship` field to register_no column (the details of the ship object represented resides in its own table, glis_ships, just the reference to the row representing that object is saved in glis_custom_ships table, which is as stated before register_no column). The details represented for this 1:n relationship are expressed by element <bag>.

Please note that in 5.8 the `Customer` mappings explained are shown, the full mapping can be found in the Appendix.

```
<hibernate-mapping>
```

```
    <class name="org.glgroup.survey.model.Customer" table="glis_customer">
        <id name="dbNumber" type="string" unsaved-value="" >
            <column name="dbnr" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
        </id>
        <property name="userId" type="string">
            <column name="userid" sql-type="varchar(32)"/>
        </property>
        <property name="password" type="string">
            <column name="passwd" sql-type="varchar(32)" />
        </property>

        <property …>
           …
        </property>

        <component name="contact" class="org.glgroup.survey.model.ContactInfo">
                            <property name="email" type="string">
                              <column name="email" sql-type="varchar(200)" />
                        </property>
          </component>
         <bag name="customerShips" table="glis_custom_ships">
              <key column="dbnr">
              </key>
              <composite-element class="org.glgroup.survey.model.CustomerShip">
                    <property name="visibility" type="string">
                        <column name="visibility" sql-type="int"/>
                    </property>
                    <property name="addresseeNumber" type="string">
                       <column name="addressee_no" sql-type="int"/>
                    </property>
                    <many-to-one name="ship" column="register_no"/>
              </composite-element>
           </bag>
    </class>
</hibernate-mapping>
```

Table 18 Hibernate mapping for Customer class

5.6.3 Class SurveyItem

The class `SurveyItem` is persisted to table gl_survey_item table which has primary key item_key column. In this class, all mappings of properties are straightforward; the fields have basic Java types, or wrapper types. The mapping structure can be seen in table.

```
<hibernate-mapping>
    <class name="org.glgroup.survey.model.SurveyItem" table="gl_survey_item">
        <!-- A 32 hex character is our surrogate key. It's automatically
            generated by Hibernate with the UUID pattern. -->
        <id name="key" type="string" unsaved-value="" >
            <column name="item_key" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
        </id>

        <property    ……>
         ……
        </property>

    </class>

</hibernate-mapping>
```

Table 19 Hibernate mapping for SurveyItem class

5.6.4 Class OrderedSurveyItem

This class is persisted to table glis_ordered_survey_items. As seen in class diagram in Figure 5-1 and more clearly in Figure 5-9, `OrderedSurveyItem` and `SurveyItem` have a 1:1 relationship. In the `OrderedSurveyItem` class, this relationship is represented with a field `item` of type `SurveyItem`. In the database tables glis_ordered_survey_items and gl_survey_item, this is represented by a foreign key to gl_survey_item table in glis_ordered_survey_items table. This foreign key is the column item_no which references the item_key column of gl_survey_item table. The element <many-to-one> represents this 1:1 relationship between the `OrderedSurveyItem` and `SurveyItem` classes.

The field `dueRange` of `OrderedSurveyItem` class is of type `DateRange`. The fields `from` and `to` of this class are persisted to item_due_on_early and item_due_on_late columns.

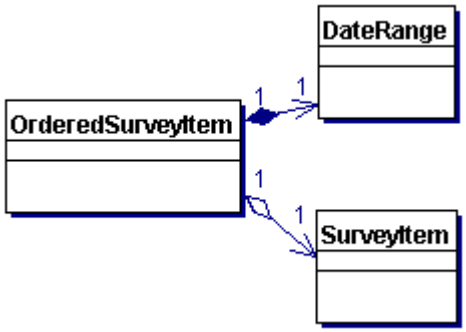

Figure 5-9 OrderedSurveyItem relationships

The remaining fields of `OrderedSurveyItem` which are not explained in this section are straightforward to map, the details will not be explained. Please note that in 5.9 the `OrderedSurveyItem` mappings explained are shown, the full mapping can be found in the Appendix.

```
<hibernate-mapping>

    <class name="org.glgroup.survey.model.OrderedSurveyItem"
table="glis_ordered_survey_items">

        <!-- A 32 hex character is our surrogate key. It's automatically
            generated by Hibernate with the UUID pattern. -->
        <id name="id" type="string" unsaved-value="" >
            <column name="id" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
        </id>

      <component name="dueRange" class="org.glgroup.survey.model.DateRange">
            <property name="from" type="date">
                  <column name="item_due_on_early" sql-type="date" />
            </property>
            <property name="to" type="date">
                  <column name="item_due_on_late" sql-type="date" />
            </property>
      </component>

      <many-to-one
            name="item"
```

```
            column="item_no"/>
    </class>

</hibernate-mapping>
```

Table 20 Hibernate mapping for OrderedSurveyItem class

5.6.5 Class OrderedSurvey

In some specific cases, an automatically mapping of Java attributes to database table columns are not possible or special handling is necessary before persisting them to database and instantiating them from the RDBMS. In these cases the Java attribute usually is of a custom type. The interface `net.sf.hibernate.UserType` should be implemented by the client programmer. A "type" class is not the actual property type - it is a class that knows how to serialize instances of another class to and from JDBC. This interface has 7 methods as can be seen in Figure 5-10.

- public int[] sqlTypes(): Return a deep copy of the persistent state, stopping at entities and at collections.
- public Object returnedClass() : The class of objects returned by nullSafeGet().
- public boolean equals(Object x, Object y) : Compare two instances of the class mapped by this type for persistence "equality"
- public Object nullSafeGet(ResultSet rs, String[] names, Object owner) throws HibernateException,SQLException : Retrieve an instance of the mapped class from a JDBC resultset
- public void nullSafeSet(PreparedStatement st, Object value, int index) throws HibernateException, SQLException: Write an instance of the mapped class to a prepared statement.

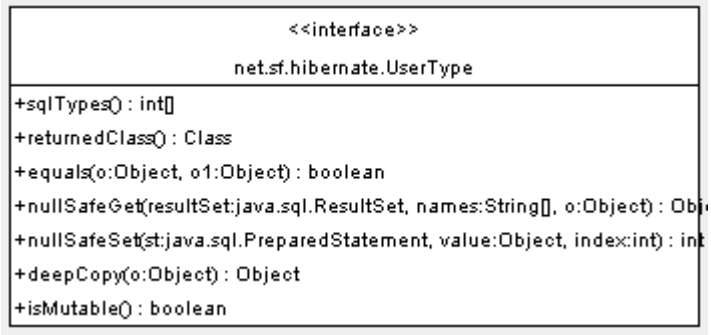

Figure 5-10 UserType interface

In the mapping for `OrderedSurvey` class, the fields `estimatedDeparture` and `estimatedArrival` must be of a custom type. The columns ETD_T and ETD_D tell the estimated departure's date and time, in the database represented as strings with a specific date and time format. In the class, we want `estimatedDeparture` field to represent date (from which one can also get the time). `estimatedDeparture` is of type `DateTime`. This class extends the class `DateTimeType` which implements the interface `net.sf.hibernate.UserType`. `DateTimeType` provides the implementations of the methods in the interface which are necessary for (manual) persistence. The columns ETD_T and ETD_D are stored /retrieved by those implemented methods in this interface. `estimatedArrival` field of `OrderedSurvey` has also type `DateTime`. It is mapped in a similar way. Figure 5-11 shows the relationships of OrderedSurvey class with other classes.

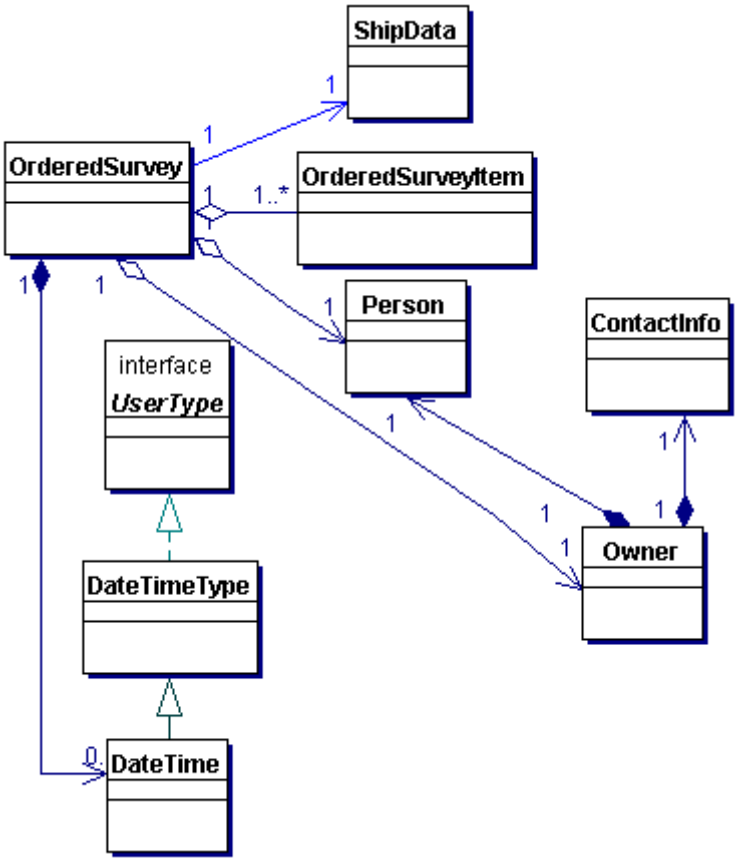

Figure 5-11 OrderedSurvey Relationships

The next very important mapping from this class is the mapping of the 1:n relationship between `OrderedSurvey` and `OrderedSurveyItem` classes. An object of type `OrderedSurvey` can have many `OrderedSurveyItem` instances. `OrderedSurvey` class has a field `orderedSurveyItems` which has its elements of type `OrderedSurveyItem`. Element <bag> in this mapping specifies the mapping for the list of ordered survey items to database tables.

There are many <component> element mappings for this class. `OrderedSurvey` has components `surveyor` and `agent` which are instances of `Person`, `vesselInfo` which is an instance of `ContactInfo` and `company` which is an instance of `Owner`.

A <many-to-one> element in the mapping maps the `ship` field of `OrderedSurvey`. `ship` is an instance of `ShipData` class, and in the glis_ordered_surveys table the column register_no is foreign key referencing register_no in glis_ships table.

Mappings of the remaining fields of `OrderedSurvey` class can be found in Appendix.

```
<hibernate-mapping>

  <class name="org.glgroup.survey.model.OrderedSurvey"
table="glis_ordered_surveys">
   <id name="orderNumber" type="string" unsaved-value="" >
            <column name="order_number" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
   </id>
```

```
    <property name="estimatedDeparture" type="org.glgroup.survey.model.DateTime">
             <column name="ETD_T" sql-type="varchar"/>
             <column name="ETD_D" sql-type="varchar"/>
    </property>

    <property name="estimatedArrival" type="org.glgroup.survey.model.DateTime">
             <column name="ETA_T" sql-type="varchar"/>
             <column name="ETA_D" sql-type="varchar"/>
    </property>

       <bag name="orderedSurveyItems" inverse="false" lazy="false">
                    <!-- order_number in glis_ordered_survey_items table-->
             <key column="order_number"/>
             <one-to-many class="org.glgroup.survey.model.OrderedSurveyItem"/>
     </bag>


        <many-to-one
                    name="ship"
                    column="register_no"/>

   <component name="surveyor" class="org.glgroup.survey.model.Person">
              <property name="name" type="string">
                    <column name="surveyor" sql-type="varchar" />
              </property>
   </component>

   <component name="agent" class="org.glgroup.survey.model.Person">
              <property name="name" type="string">
                    <column name="agent" sql-type="varchar" />
              </property>
         <component name="contact" class="org.glgroup.survey.model.ContactInfo">
                           <property name="phone" type="string">
                             <column name="agent_tel" sql-type="varchar" />
                      </property>
                     <property name="fax" type="string">
                             <column name="agent_fax" sql-type="varchar" />
                      </property>
              </component>
            </component>

            <component name="vesselInfo"
class="org.glgroup.survey.model.ContactInfo">
                    <property name="phone" type="string">
                          <column name="vessel_tel" sql-type="varchar" />
                  </property>
            </component>

       <component name="company" class="org.glgroup.survey.model.Owner">
              <property name="companyName" type="string">
                    <column name="company" sql-type="varchar" />
              </property>
         <component name="contact" class="org.glgroup.survey.model.ContactInfo">
                      <property name="email" type="string">
                             <column name="email" sql-type="varchar" />
                      </property>
          </component>

           <component name="contactPerson" class="org.glgroup.survey.model.Person">
              <property name="name" type="string">
                    <column name="name" sql-type="varchar" />
              </property>
             <component name="contact" class="org.glgroup.survey.model.ContactInfo">
                           <property name="phone" type="string">
                             <column name="tel" sql-type="varchar" />
                      </property>
                     <property name="fax" type="string">
                             <column name="fax" sql-type="varchar" />
                      </property>
```

```
            </component>
          </component>
        </component>
    </class>

</hibernate-mapping>
```

Table 21 Hibernate mapping for OrderedSurvey class

5.7 Detailed Sequence Diagrams

The interaction between the components of the system can be seen in the following sequence diagrams in
Figure 5-12 and Figure 5-13. In
Figure 5-12, login in of a user to the system is shown, and in the second viewing the survey request for a specific ship.

For a log in to GL Fleet Online web application, a client initiates a logon request. The Logon request is the logon action in Struts, so the controller dispatches to `LogonAction` class. In the `LogonAction` class, the `execute(X)` method has a reference to the form bean which this class uses, i.e., `LogonOXFForm` bean. `LogonOXFForm` bean contains the values from the HTML form which is sent with the user request. It contains username and password fields. In the `LogonAction` class, this username and password of the user are retrieved, and these values are passed to Hibernate query which is used to select the customer object with the given username and password from the database. If the query is successful, the customer object is returned to the `LogonAction`. `LogonAction` saves the customer object in the HTTPSession for this customer. The next step in the interaction after successful login is forwarding the request to Struts' `ShowFleetAction` class. `ShowFleetAction` retrieves the ships and related surveys for the ships from the database by using Hibernate queries for the customer session. Then Struts `ShowFleetAction` forwards the request to OXF ShowFleetStatus pipeline. This pipeline creates the HTML page that will be sent to the user. The HTML contains the fleet status of the ships of customer. When Struts `ShowFleetAction` forwards the request to ShowFleetStatus, the OXF Processor gets the request. It selects the corresponding pipeline, and processes it. As a result the pipeline ShowFleetStatus.xpl creates the output page.

The next interaction of the components to be shown is when the client requests to view the details of a specific survey for a ship. In this interaction sequence, the client initiates by request to view the survey status for a specific ship. Struts controller gets the request and dispatches it to `ShowSurveyRequestAction`. It checks the session object saved for the customer. It retrieves the selected survey chosen by customer. It stores this survey in HTTPSession object for the customer and forwards the request to ShowSurveyRequest pipeline. OXF Processor gets the forward and executes the pipeline. As a result of this pipeline the HTML page showing the details for the chosen survey is sent back to client. This interaction can be seen in Figure 5-13.

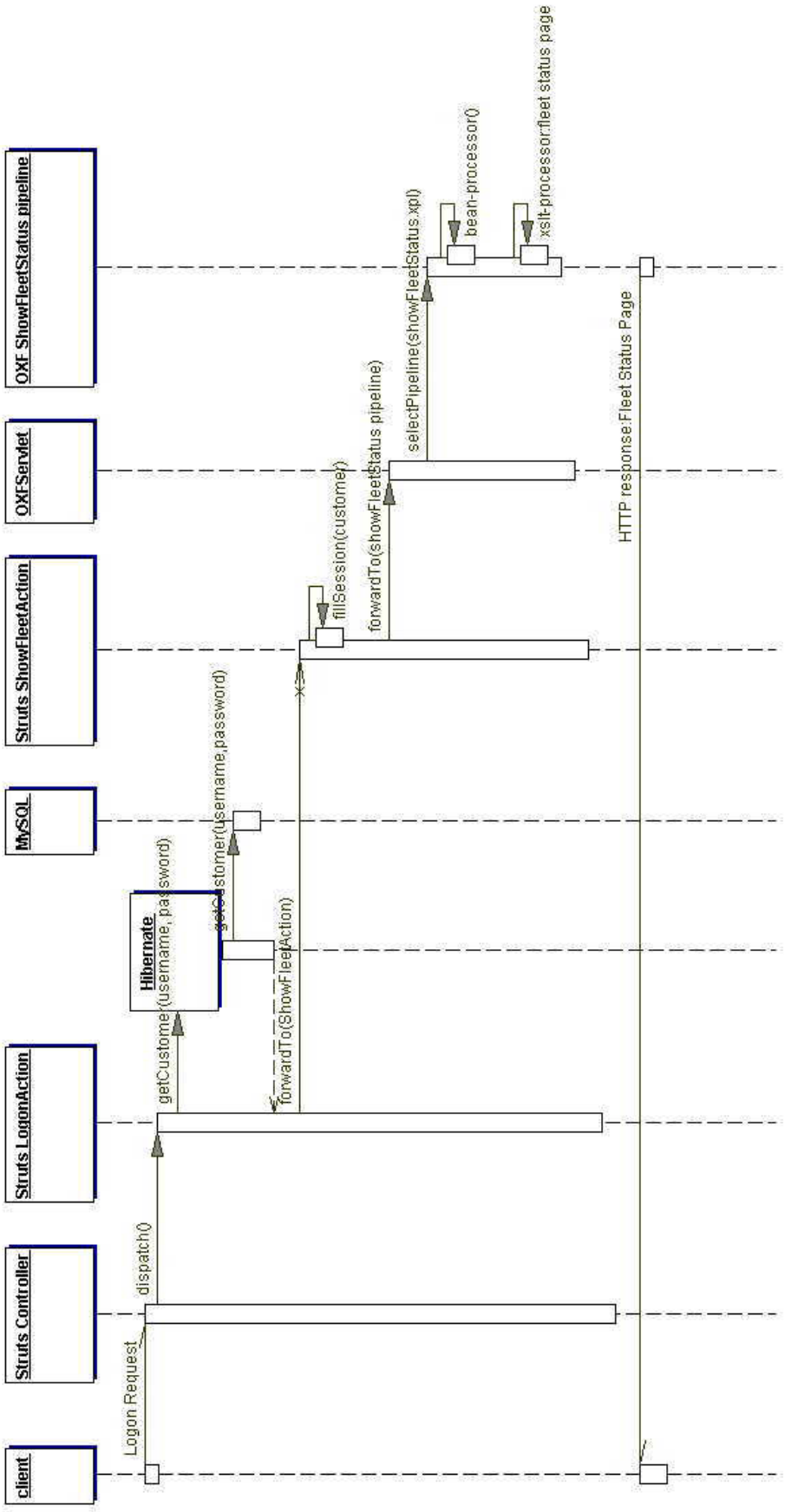

Figure 5-12 Interaction Diagram for Login to Fleet Online

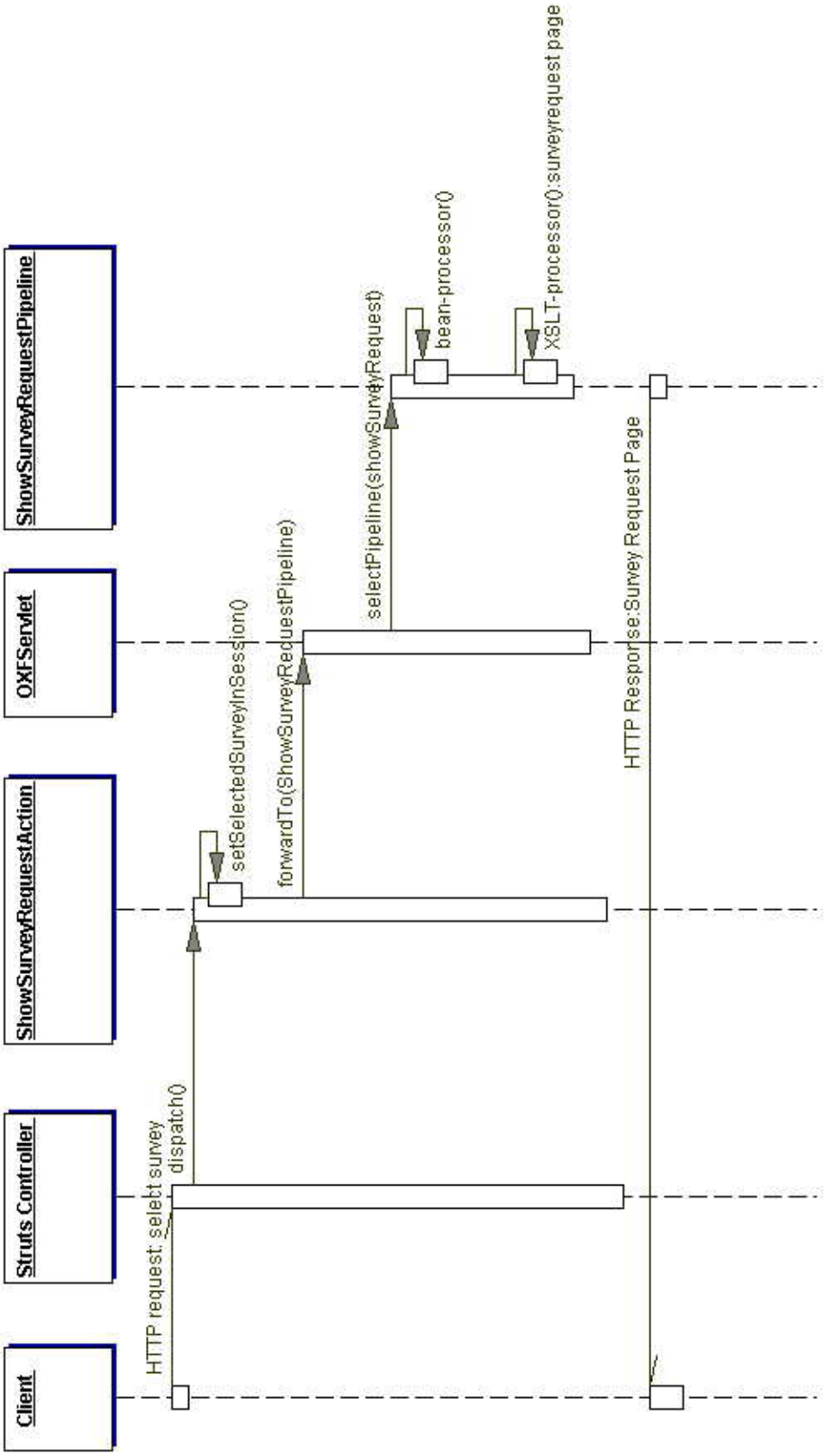

Figure 5-13 Interaction Diagram for viewing survey request

5.8 Screenshots

The GL Fleet Online web application has five main views. At the entrance to the site, currently a simple page is shown with a link to logon page. Figure 5-14 shows this page. This page is created by a JSP page, index.jsp.

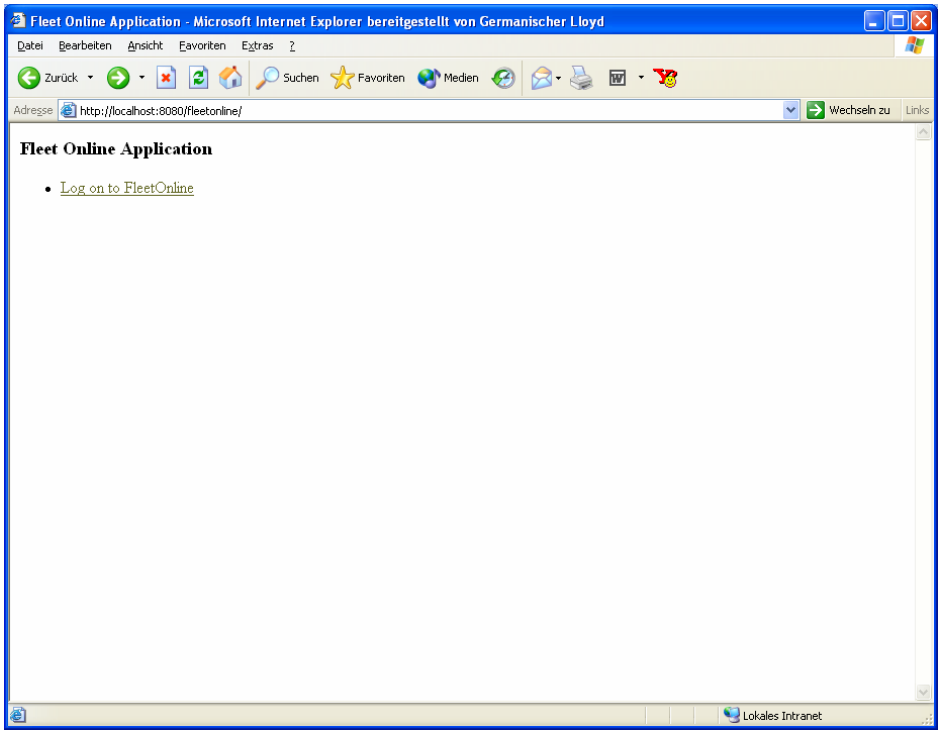

Figure 5-14 Main page-index.jsp output

When the user clicks the link “Log on to FleetOnline” seen in Figure 5-14, the request is forwarded to login page, and the user views the logon page as response. The logon page can be seen in Figure 5-15 Logon Page. After the user enters his/her username and password, he/she sends the HTTP request to the application. If the login is successful, the page with fleet status for his/her ships is displayed to him.

The fleet status page shown in Figure 5-16 shows mainly short overview information about the existing ships for the customer and the survey requests for each ship. This page is created by OXF fleet status pipeline. The customer can select one of the survey requests by clicking and the details of the survey request will be sent by the application as HTTP response. The survey request page sent for a specific survey request can be seen in Figure 5-17.

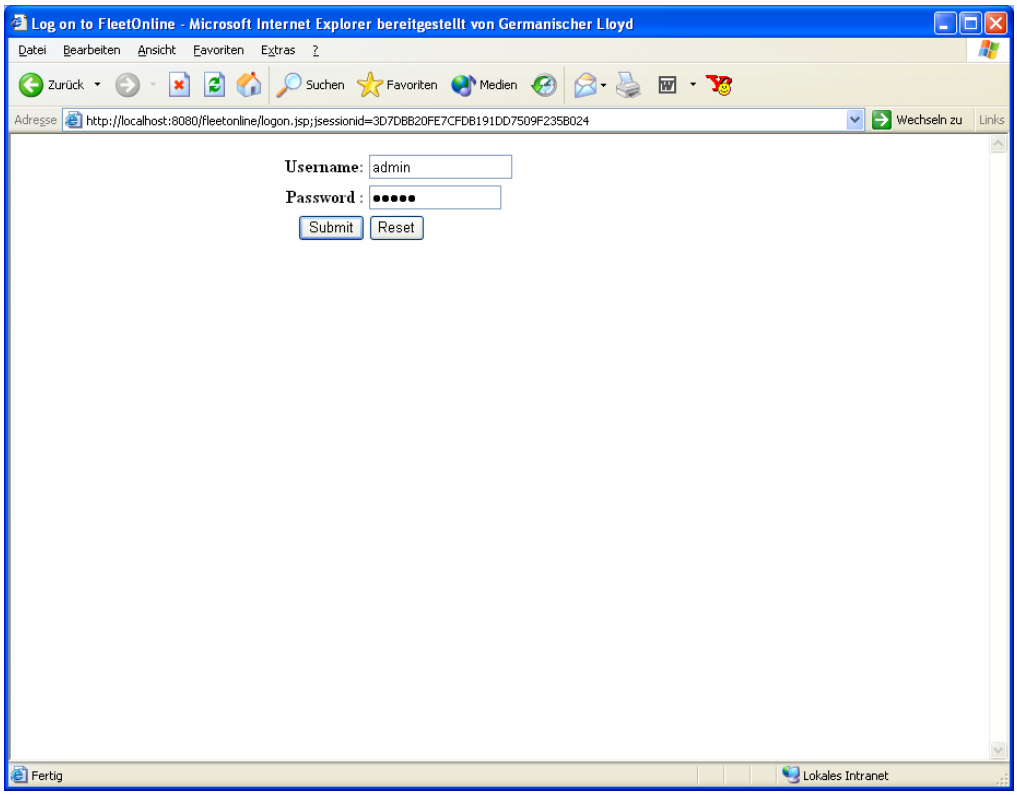

Figure 5-15 Logon Page

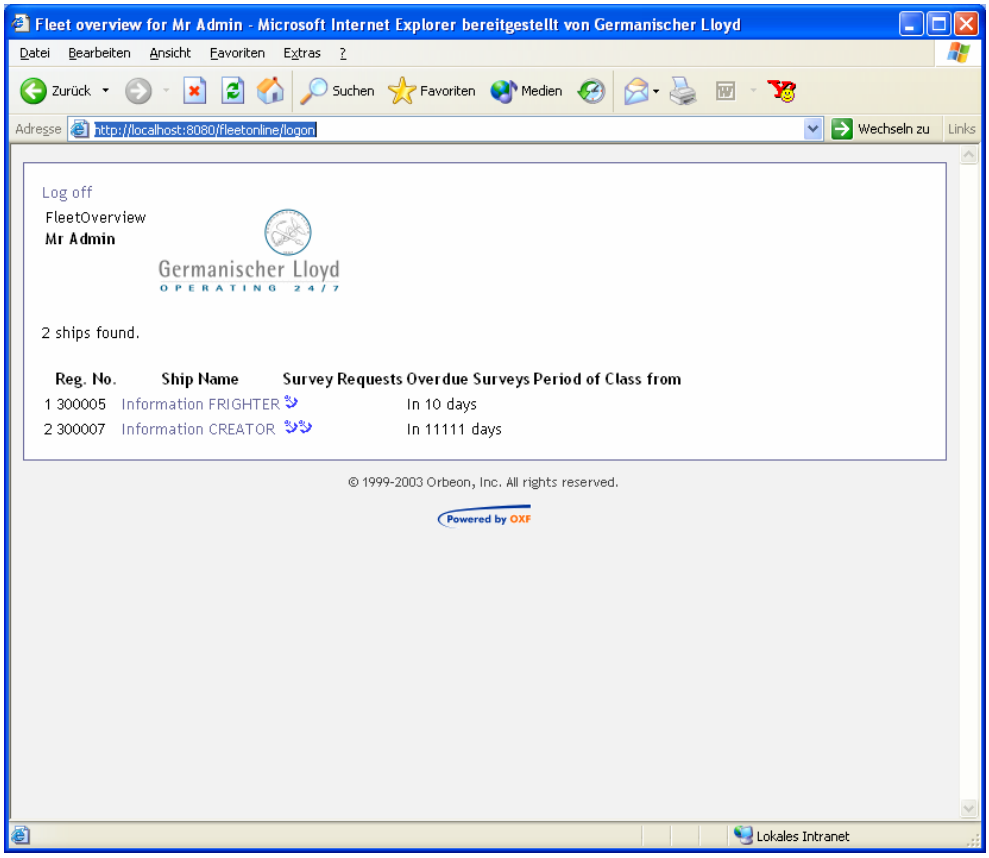

Figure 5-16 Fleet status page for a specific customer

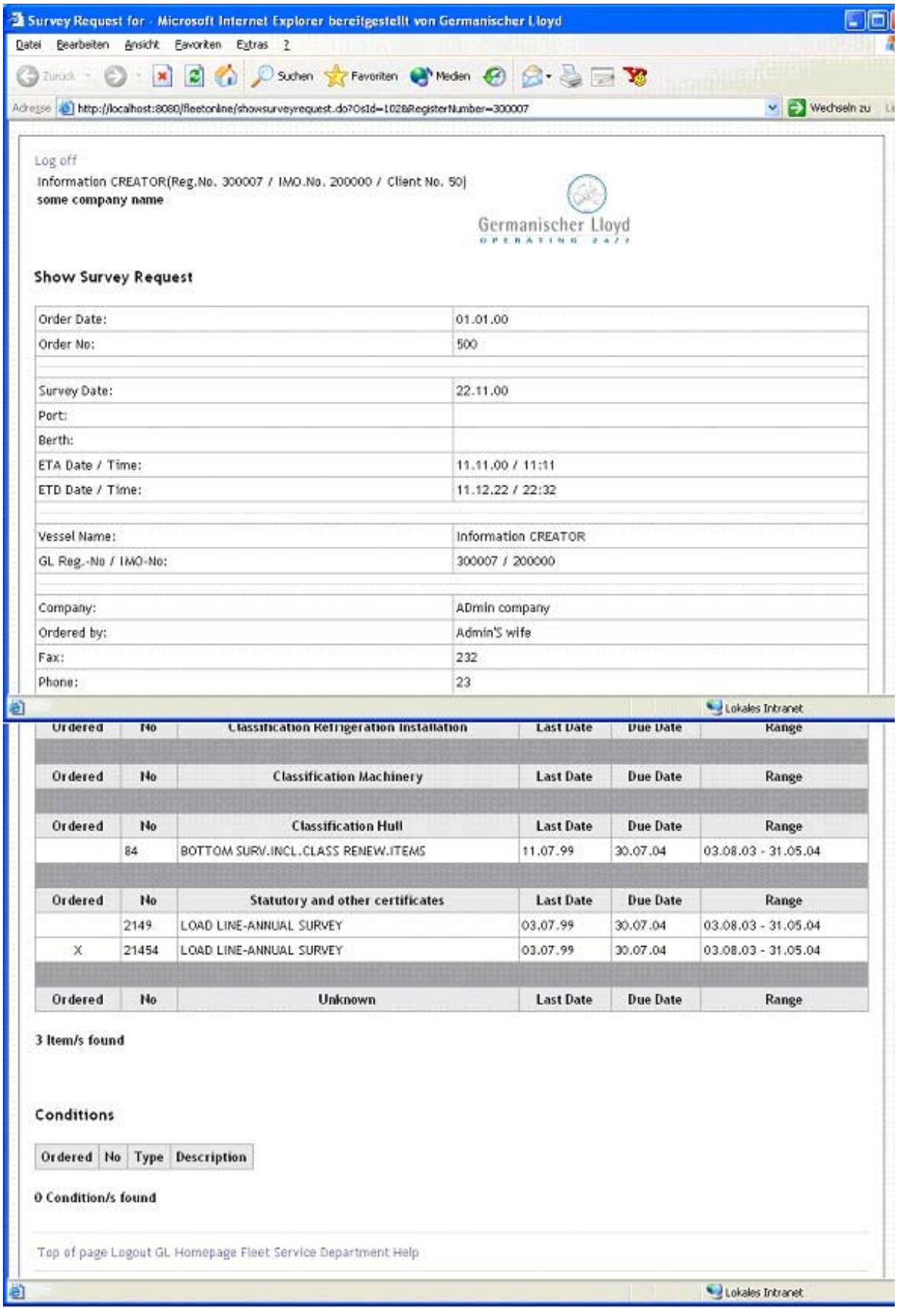

Figure 5-17 Survey request page

6. Conclusion

6.1 Summary

Web application development frameworks make the life easier for web development teams. The existing frameworks Struts, Cocoon and OXF have been analysed and evaluated in the work. As a result of the analysis, Struts-OXF integration framework was chosen. Struts is an MVC-based framework, the model, view and controller in Struts are components easy to understand. The documentation and very large user base and efficiency of Struts make it a very suitable framework for most web development projects.

OXF is an XML content-based J2EE application development framework. OXF has very useful concepts like XML pipelines. The company offering OXF also offers an OXF-Struts integration framework. In this integration framework Struts functionalities as well as main OXF functionalities are provided to the application programmer. The concept of XML pipelines and view creation by applying XSL transformations to XML documents are introduced to Struts. In pure Struts, the views are JSPs. By OXF a new view creation concept is introduced.

Object-relational mappers ease the mapping of business objects to RDBMS. This task is not a simple task, because of the different views on data structures in both worlds. Object-oriented design and languages have concepts like inheritance, polymorphism, classes which do not exist in relational tables. There are many existing patterns to map the OO classes to relational tables.

The analysed O/R mapper frameworks in this student project work are Hibernate, OJB (Object-relational Bridge) and Castor JDO. After analysis, Hibernate is chosen for O/R mapping. Hibernate is a high-performance O/R mapper for Java with very large user base, very nice documentation. Hibernate has many features enabling flexible mapping of classes to relational tables. But it can be said that Hibernate is mostly suitable for mapping existing Java classes to relational tables. Hibernate provides tools to create the relational tables from the mapping files of the Java classes.

GL Fleet Online web application which provides online facilities for GL customers is prototypically implemented with chosen web development framework and O/R mapper. GL Fleet Online application provides ship owners to inspect the various data about their ships. The ships should have surveys in specified times. The online application mainly provides the functionality to view the fleet status and survey request details. For the domain model persistence, Hibernate has been used.

6.2 Possible further development

The GL Fleet Online service for customers includes many tasks. Some further development points in this work can be:

- New requirements
- New Design of GL Fleet Online System database schema
- Session management
- Move to Oracle database, the main GL database

GL Fleet Online System offers GL's customers much functionality. In the project work only some part of the requirements were implemented. Use cases like registering of new customers, addition of new ships, processing of survey request data etc. have not been implemented in this work. The addition of the completing use cases is a main development point.

The current GL Fleet Online tables are not expressing the business object structures very well. The tables have missing constraints which can be only seen at application logic level. The fact that the tables were already being used by existing GL applications made it difficult to change the existing tables. Other than that it was not desired by GL Fleet Online team to change the tables. So, in my work other than the very necessary cases, the tables are not changed, only the type definitions are matched with MySQL. There are some minor SQL type differences between Oracle DB and MySQL DB. The overall new design of the Fleet Online database schema is a very important development point.

After logon, necessary customer data for session management are held in customer's session object. This object is saved in a session. This object holds all the customer data with his ships and survey requests data. When the customer has many ships with many survey requests whose data size is big, the session object will be very big in size. Because this object will be residing in memory until the customer logs off, it will be additional load (memory usage) for the Fleet Online server. The future work is to design the class so that the session object for customer holds only ids for customer, ships and survey requests, not the data itself. When the data with specific id is requested, then it will be retrieved from database or any other resource where the data resides.

The last further development point is moving to Oracle database. The implementation and mappings done for this work have been tested with MySQL database. There is a need to move to GL's main database. There will be little changes in mapping file for Hibernate because of SQL type differences between Oracle and MySQL.

Appendix

A1. Web.xml deployment descriptor

```
<?xml version="1.0" ?>
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>

     <context-param>
         <param-name>oxf.resources.factory</param-name>
         <param-
value>org.orbeon.oxf.resources.PriorityResourceManagerFactory</param-value>
     </context-param>
     <context-param>
         <param-name>oxf.resources.webapp.rootdir</param-name>
         <param-value>/WEB-INF/resources</param-value>
     </context-param>
     <context-param>
         <param-name>oxf.resources.priority.1</param-name>
         <param-value>org.orbeon.oxf.resources.WebAppResourceManagerFactory</param-
value>
     </context-param>
     <context-param>
         <param-name>oxf.resources.priority.2</param-name>
         <param-
value>org.orbeon.oxf.resources.ClassLoaderResourceManagerFactory</param-value>
     </context-param>

     <context-param>
         <param-name>oxf.properties</param-name>
         <param-value>oxf:/config/properties.xml</param-value>
     </context-param>

 <servlet>
    <servlet-name>testservlet</servlet-name>
    <servlet-class>org.glgroup.hibernate.mapping.TestServlet</servlet-class>
   <init-param>
      <param-name>load-class</param-name>
      <param-value>
        com.mysql.jdbc.Driver
      </param-value>
    </init-param>
  <load-on-startup>1</load-on-startup>
  </servlet>

    <servlet>
        <servlet-name>struts</servlet-name>
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>10</param-value>
        </init-param>
        <init-param>
            <param-name>mapping</param-name>
            <param-value>org.apache.struts.action.RequestActionMapping</param-
value>
        </init-param>
        <!-- this package doesnt exist any more, deleted
```

```
        <init-param>
            <param-name>application</param-name>
            <param-
value>org.orbeon.oxf.struts.examples.ApplicationResources</param-value>
        </init-param>
        -->
        <init-param>
            <param-name>locale</param-name>
            <param-value>true</param-value>
        </init-param>

        <init-param>
       <param-name>load-class</param-name>
             <param-value>
         com.mysql.jdbc.Driver
             </param-value>
       </init-param>

      <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet>
        <servlet-name>oxf</servlet-name>
        <servlet-class>org.orbeon.oxf.servlet.ProcessorServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>



<!-- Action Servlet Mapping -->



  <servlet-mapping>
    <servlet-name>testservlet</servlet-name>
    <url-pattern>/testing</url-pattern>
  </servlet-mapping>

   <servlet-mapping>
        <servlet-name>struts</servlet-name>
        <url-pattern>/</url-pattern>
   </servlet-mapping>

  <!--
  <servlet-mapping>
    <servlet-name>struts</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  -->

   <servlet-mapping>
        <servlet-name>oxf</servlet-name>
        <url-pattern>*.xpl</url-pattern>
   </servlet-mapping>

   <servlet-mapping>
        <servlet-name>oxf</servlet-name>
        <url-pattern>*.css</url-pattern>
   </servlet-mapping>
   <servlet-mapping>
        <servlet-name>oxf</servlet-name>
        <url-pattern>*.js</url-pattern>
   </servlet-mapping>

   <servlet-mapping>
        <servlet-name>oxf</servlet-name>
        <url-pattern>*.gif</url-pattern>
   </servlet-mapping>
```

```
   <servlet-mapping>
        <servlet-name>oxf</servlet-name>
        <url-pattern>*.png</url-pattern>
   </servlet-mapping>


<!-- Application Tag Library Descriptor -->
  <taglib>
    <taglib-uri>/WEB-INF/app.tld</taglib-uri>
    <taglib-location>/WEB-INF/app.tld</taglib-location>
  </taglib>

  <!-- Struts Tag Library Descriptors -->
  <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
  </taglib>

<resource-ref>
      <res-ref-name>jdbc/fleetonline</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
   </resource-ref>
</web-app>
```

A2. Struts configuration file

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
  "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>
    <form-beans>

      <form-bean name="logonOXFForm"
type="org.glgroup.fleetonline.struts.LogonOXFForm"/>

</form-beans>

<global-forwards>
      <forward   name="mainpage"                path="/index.jsp"/>
    <forward   name="logon"                 path="/logon.jsp"/>
    <forward   name="logoff"                 path="/logoff.do"/>
    <forward   name="showFleet"           path="/ShowFleet.xpl"/>
    <forward   name="showFleetStruts"           path="/showfleet.do"/>
    <forward   name="showSurveyRequestStruts"
path="/showsurveyrequest.do"/>
    <forward   name="showSurveyRequest"    path="/ShowSurveyRequest.xpl"/>
  </global-forwards>
    <action-mappings>
```

```
        <action path="/" forward="/index.jsp"/>
      <action    path="/logon"
                type="org.glgroup.fleetonline.struts.LogonAction"
                name="logonOXFForm"
               scope="request"
               input="logon.jsp">
      <forward name="success" path="/logon.jsp"/>
      </action>

       <!-- Process a user logoff -->
    <action    path="/logoff"
               type="org.glgroup.fleetonline.struts.LogoffAction">
              <forward name="success" path="/index.jsp"/>
    </action>

      <action    path="/showfleet"
                type="org.glgroup.fleetonline.struts.ShowFleetAction">
              <forward name="success" path="/ShowFleet.xpl"/>
    </action>

    <action  path="/showsurveyrequest"

type="org.glgroup.fleetonline.struts.ShowSurveyRequestAction">
              <forward name="success" path="/ShowSurveyRequest.xpl"/>
    </action>

    </action-mappings>
   <message-resources
    parameter="org.glgroup.fleetonline.struts.ApplicationResources"/>

  <message-resources

parameter="org.glgroup.fleetonline.struts.AlternateApplicationResources"
    key="alternate">
  </message-resources>

<!-- this does not exist any more
<message-resources
parameter="org.orbeon.oxf.struts.examples.ApplicationResources"
        null="false" key="messages" />
-->
    <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
        <set-property property="pathnames" value="/WEB-INF/validator-
rules.xml,
                                                  /WEB-
INF/validation.xml"/>
    </plug-in>
</struts-config>
```

A3. Hibernate mapping files

A3.1 Ship.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
```

```
<hibernate-mapping>

    <class name="org.glgroup.survey.model.ShipData" table="glis_ships">

        <id name="registerNumber" type="string" unsaved-value="" >
            <column name="register_no" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
        </id>
        <property name="name" type="string">
                <column name="name" sql-type="varchar" />
        </property>

        <component name="classPeriod"
class="org.glgroup.survey.model.DateRange">
                <property name="from" type="date">
                      <column name="ship_in_class" sql-type="date"/>
                </property>
          </component>

        <component name="dueRemaining"
class="org.glgroup.survey.model.DueRemaining">
              <property name="overdue" type="boolean">
                <column name="overdue" sql-type="int(1)" />
          </property>
            <property name="deadline" type="string">
                <column name="deadline" sql-type="int" />
            </property>
            </component>
            <property name="suspended" type="boolean">
                <column name="suspended" sql-type="int(1)" />
        </property>

        <property name="csItems" type="string">
                <column name="csitems" sql-type="int" />
        </property>
        <property name="imoNumber" type="string">
                <column name="imo_no" sql-type="varchar" />
        </property>
        <property name="flagId" type="string">
                <column name="flag_id" sql-type="int(6)" />
        </property>
    </class>

</hibernate-mapping>
```

A3.2 Customer.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>

    <class name="org.glgroup.survey.model.Customer" table="glis_customer">
        <id name="dbNumber" type="string" unsaved-value="" >
            <column name="dbnr" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
        </id>
```

```
        <property name="userId" type="string">
            <column name="userid" sql-type="varchar(32)"/>
        </property>
        <property name="password" type="string">
            <column name="passwd" sql-type="varchar(32)" />
        </property>

            <property name="accessRights" type="string">
            <column name="rights" sql-type="int" />
        </property>
            <property name="sidgen" type="string">
            <column name="sidgen" sql-type="int" />
        </property>
        <property name="sid" type="string">
            <column name="sid" sql-type="int" />
        </property>
        <property name="name" type="string">
            <column name="name" sql-type="varchar(50)" />
        </property>
        <property name="logtime" type="string">
            <column name="logtime" sql-type="int" />
        </property>

        <property name="lastAccess" type="date">
            <column name="last_access" sql-type="date" />
        </property>

        <property name="accessCounter" type="int">
            <column name="access_counter" sql-type="int" />
        </property>

        <property name="passwordChanged" type="date">
            <column name="passwd_changed" sql-type="date" />
        </property>
        <property name="shipsAccess" type="date">
            <column name="ships_access" sql-type="date"/>
        </property>

        <property name="overdueItems" type="boolean">
            <column name="overdue" sql-type="int(1)"/>
        </property>
        <property name="shipNumber" type="int">
            <column name="ships_number" sql-type="int"/>
        </property>
        <property name="shipsValid" type="boolean">
            <column name="ships_valid" sql-type="int"/>
        </property>
        <property name="suspended" type="boolean">
            <column name="suspended" sql-type="int(1)"/>
        </property>
        <property name="lockUnlockDate" type="date">
            <column name="lock_unlock_date" sql-type="date"/>
        </property>
        <property name="accountLocked" type="boolean">
            <column name="account_locked" sql-type="int"/>
        </property>
        <component name="contact"
class="org.glgroup.survey.model.ContactInfo">
                        <property name="email" type="string">
                          <column name="email" sql-type="varchar(200)" />
                    </property>
          </component>
```

```
          <bag name="customerShips" table="glis_custom_ships">
            <key column="dbnr">
            </key>
            <composite-element
class="org.glgroup.survey.model.CustomerShip">
                  <property name="visibility" type="string">
                      <column name="visibility" sql-type="int"/>
                    </property>
                    <property name="addresseeNumber" type="string">
                      <column name="addressee_no" sql-type="int"/>
                    </property>
                    <many-to-one name="ship" column="register_no"/>
            </composite-element>
          </bag>

    </class>

</hibernate-mapping>
```

A3.3 SurveyItem.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>

    <class name="org.glgroup.survey.model.SurveyItem"
table="gl_survey_item">

        <!-- A 32 hex character is our surrogate key. It's automatically
            generated by Hibernate with the UUID pattern. -->
        <id name="key" type="string" unsaved-value="" >
            <column name="item_key" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
        </id>

        <property name="comment" type="string">
            <column name="comment" sql-type="text" />
        </property>
        <property name="description" type="string">
            <column name="description" sql-type="varchar(100)" />
        </property>
        <property name="name" type="string">
            <column name="name" sql-type="varchar(32)" />
        </property>
        <property name="chapterKey" type="string">
            <column name="chapter_key" sql-type="varchar(20)" />
        </property>
        <property name="type" type="string">
            <column name="item_type" sql-type="varchar(1)" />
        </property>
    </class>

</hibernate-mapping>
```

A3.4 OrderedSurveyItem.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>

    <class name="org.glgroup.survey.model.OrderedSurveyItem"
table="glis_ordered_survey_items">

        <!-- A 32 hex character is our surrogate key. It's automatically
            generated by Hibernate with the UUID pattern. -->
        <id name="id" type="string" unsaved-value="" >
            <column name="id" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
        </id>

        <property name="orderNumber" type="string">
            <column name="order_number" sql-type="int"/>
        </property>
        <property name="lastDate">
            <column name="item_event_on" sql-type="date" />
        </property>

        <property name="dueDate">
            <column name="item_due_on" sql-type="date" />
        </property>
        <property name="checked" type="boolean">
            <column name="checked" sql-type="int" />
        </property>
        <property name="state" type="string">
            <column name="item_state" sql-type="int" />
        </property>


        <component name="dueRange"
class="org.glgroup.survey.model.DateRange">
            <property name="from" type="date">
                  <column name="item_due_on_early" sql-type="date" />
            </property>
            <property name="to" type="date">
                  <column name="item_due_on_late" sql-type="date" />
            </property>
        </component>
            <many-to-one
                  name="item"
                  column="item_no"/>

    </class>

</hibernate-mapping>
```

A3.5 OrderedSurvey.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
                                   "http://hibernate.sourceforge.net/hibernate-
mapping-2.0.dtd">

<hibernate-mapping>

    <class name="org.glgroup.survey.model.OrderedSurvey"
table="glis_ordered_surveys">

        <!-- A 32 hex character is our surrogate key. It's automatically
            generated by Hibernate with the UUID pattern. -->
        <id name="orderNumber" type="string" unsaved-value="" >
            <column name="order_number" sql-type="int" not-null="true"/>
            <generator class="assigned"/>
        </id>

        <property name="estimatedDeparture"
type="org.glgroup.survey.model.DateTime">
            <column name="ETD_T" sql-type="varchar"/>
            <column name="ETD_D" sql-type="varchar"/>
        </property>

         <property name="estimatedArrival"
type="org.glgroup.survey.model.DateTime">
            <column name="ETA_T" sql-type="varchar"/>
            <column name="ETA_D" sql-type="varchar"/>
        </property>


        <property name="dbNumber" type="string">
            <column name="dbnr" sql-type="int"/>
        </property>
        <property name="userType" type="string">
            <column name="userType" sql-type="int" />
        </property>

             <property name="orderDate" type="date">
            <column name="order_date" sql-type="date" />
        </property>
             <property name="surveyDate" type="date">
            <column name="survey_date" sql-type="date" />
        </property>

        <property name="orderConfirmation" type="string">
            <column name="order_confirmation" sql-type="int" />
        </property>

        <property name="portOfSurvey" type="string">
            <column name="port_of_survey" sql-type="text" />
        </property>

        <property name="berthOfSurvey" type="string">
            <column name="berth_of_survey" sql-type="text" />
        </property>

        <property name="surveyState" type="string">
            <column name="survey_state" sql-type="int" />
        </property>

        <property name="remarks" type="string">
            <column name="remarks" sql-type="text" />
        </property>
```

```
        <property name="emailConfirmation" type="boolean">
            <column name="email_confirmation" sql-type="int" />
        </property>

             <!--
             <property name="orderedSurveyItemList"
type="org.glgroup.survey.model.OrderedSurveyItemList">
            <column name="order_number" sql-type="int" />
        </property>
             -->

        <component name="surveyor" class="org.glgroup.survey.model.Person">
             <property name="name" type="string">
                    <column name="surveyor" sql-type="varchar" />
             </property>
             <!--
                    <property name="firstName" type="string">
                    <column name="???" sql-type="varchar" />
             </property>

             <component name="contact"
class="org.glgroup.survey.model.ContactInfo">
                          <property name="phone" type="string">
                            <column name="tel" sql-type="varchar" />
                      </property>
                     <property name="fax" type="string">
                            <column name="fax" sql-type="varchar" />
                      </property>

                    <property name="email" type="string">
                      <column name="email" sql-type="text" />
                          </property>
             </component>
             -->
          </component>

          <component name="agent" class="org.glgroup.survey.model.Person">
             <property name="name" type="string">
                    <column name="agent" sql-type="varchar" />
             </property>
                    <!--
                    <property name="firstName" type="string">
                    <column name="????" sql-type="varchar" />
             </property>
             -->
             <component name="contact"
class="org.glgroup.survey.model.ContactInfo">
                           <property name="phone" type="string">
                            <column name="agent_tel" sql-type="varchar" />
                      </property>
                     <property name="fax" type="string">
                            <column name="agent_fax" sql-type="varchar" />
                      </property>
                    <!--
                    <property name="email" type="string">
                      <column name="????" sql-type="varchar" />
                          </property>
                          -->
             </component>
          </component>

          <component name="vesselInfo"
class="org.glgroup.survey.model.ContactInfo">
                    <property name="phone" type="string">
                          <column name="vessel_tel" sql-type="varchar" />
                </property>
                <!--
             <property name="fax" type="string">
```

```
                        <column name="???" sql-type="varchar" />
                </property>
                -->
          </component>

          <component name="company" class="org.glgroup.survey.model.Owner">
             <property name="companyName" type="string">
                    <column name="company" sql-type="varchar" />
             </property>
             <component name="contact"
class="org.glgroup.survey.model.ContactInfo">
                          <!--
                           <property name="phone" type="string">
                             <column name="???" sql-type="varchar" />
                      </property>
                     <property name="fax" type="string">
                             <column name="???" sql-type="varchar" />
                      </property>
                      -->
                      <property name="email" type="string">
                             <column name="email" sql-type="varchar" />
                      </property>
             </component>


             <!--ORDERING PERSON INFO-->
             <component name="contactPerson"
class="org.glgroup.survey.model.Person">
             <property name="name" type="string">
                    <column name="name" sql-type="varchar" />
             </property>
             <!--
                    <property name="firstName" type="string">
                    <column name="???" sql-type="varchar" />
             </property>
             -->
             <component name="contact"
class="org.glgroup.survey.model.ContactInfo">
                           <property name="phone" type="string">
                             <column name="tel" sql-type="varchar" />
                      </property>
                     <property name="fax" type="string">
                             <column name="fax" sql-type="varchar" />
                      </property>
                      <!--
                    <property name="email" type="string">
                      <column name="email" sql-type="text" />
                          </property>
                          -->
             </component>
          </component>
          </component>

            <many-to-one
                    name="ship"
                    column="register_no"/>

             <bag name="orderedSurveyItems" inverse="false" lazy="false">
                    <!-- order_number in glis_ordered_survey_items table-->
            <key column="order_number"/>
            <one-to-many class="org.glgroup.survey.model.OrderedSurveyItem"/>
        </bag>

    </class>

</hibernate-mapping>
```

References

[CJDO03] Castor JDO framework
http://castor.exolab.org/jdo.html/; 2003

[CGI03] Common Gateway Interface
http://www.w3.org/CGI/; 2003

[COCO03] Cocoon Framework
http://cocoon.apache.org/2.1/; 2003

[CSS03] Cascading Style Sheets
http://www.w3.org/Style/CSS/; 2003

[CoCoN03] The Apache Cocoon Project
http://cocoon.apache.org/; 2003

[CoPool03] The Apache Jakarta Commons Pool Component
http://jakarta.apache.org/commons/pool/; 2003

[CRAIG03] Craig McClanahan, Building Web Applications with the Struts Framework
http://www.apache.org/~craigmcc/apachecon-2002-struts.ppt; 2003

[DBCP03] The Apache Jakarta Commons DBCP Component
http://jakarta.apache.org/commons/dbcp/; 2003

[EJB03] Enterprise JavaBeans Technology
http://java.sun.com/products/ejb/index.jsp; 2003

[Fowler03] Patterns of Enterprise Application Architecture
Martin Fowler; March 2003

[FTEAM03] The Germanischer Lloyd Fleet Online Team
https://www.gl-group.com/fleetonline/login_team.html; 2003

[GHJV95] Design Patterns
by Erich Gamma (Author), Richard Helm (Author), Ralph Johnson (Author), John Vlissides (Author); 1995

[GL03] Germanischer Lloyd Homepage
http://www.gl-group.com; 2003

[GLv03] GL description briefly
http://www.gl-group.com/cgi-bin/w/w3red?SET=HOME&SUB=1; 2003

[HBNT03] Hibernate Object Relational Mapping Framework
http://www.hibernate.org/; 2003

[HibeRef03] Hibernate, O/R mapper framework for Java, documentation
http://www.hibernate.org/hib_docs/reference/html/; 2003

[HIBEa1] Introduction to Hibernate
www.systemmobile.com/articles/hibernate_intro.php; 2003

[HTML03] HTML Technology
http://www.w3.org/MarkUp/; 2003

[HTTP03] Hypertext Transfer Protocol
http://www.w3.org/Protocols/; 2003

[IBMSTR03] Struts, an open-source MVC implementation
http://www-106.ibm.com/developerworks/library/j-struts/?dwzone=java; 2003

[J2EE03] Java 2 Platform, Enterprise Edition
http://java.sun.com/j2ee/; 2003

[Java03] Java Technology
http://java.sun.com/; 2003

[JaXP03] Java API for XML Processing
http://java.sun.com/xml/jaxp/; 2003

[JDBC03] JDBC Technology
http://java.sun.com/products/jdbc/;2003

[JDO03] Sun Java Data Objects
http://java.sun.com/products/jdo/; 2003

[JnDI03] Java Naming and Directory Interface
http://java.sun.com/products/jndi/; 2003

[JSP03] Java Server Pages Technology
http://java.sun.com/products/jsp/; 2003

[JSPv03] JSP Technology Overview
http://java.sun.com/products/jsp/overview.html; 2003

[JSTL03] JavaServer Pages Standard Tag Library
http://java.sun.com/products/jsp/jstl/index.jsp; 2003

[SerJSPa03] A tutorial on Java Servlets and Java Server Pages
http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/; 2003

[JtA03] Java Transaction API
http://java.sun.com/products/jta/; 2003

[LCoCoN03] Learn to create a simple Cocoon sitemap
http://builder.com.com/5100-6387-5055241.html ;2003

[MySQL03] MySQL Database Server
http://www.mysql.com/; 2003

[ODMG03] Object Data Management Group, The Standard for Storing Objects
http://www.odmg.org/; 2003

[OJB03] The Apache DB Project : Object Relational Bridge
http://db.apache.org/ojb/; 2003

[OJBa1] OJB presentation
www.ajug.org/meetings/download/ojb.pdf;2003

[ORBE] OXF User Guide
http://www.orbeon.com/oxf/doc/; 2003

[ORcL9i] Oracle Database
http://www.oracle.com/ip/deploy/database/oracle9i/;2003

[ORTC03] Object Relational Tool Comparison
http://c2.com/cgi-bin/wiki?ObjectRelationalToolComparison; 2003

[OXF03] OXF-The XML Platform
http://www.orbeon.com/oxf/; 2003

[OxfRef03] OXF- The XML Platform Official Documentation
http://www.orbeon.com/oxf/doc/; 2003

[OxfAPI03] OXF Processor Java API
http://www.orbeon.com/oxf/doc/reference-processor-api; 2003

[SaX03] Simple API for XML
http://www.saxproject.org/; 2003

[Servlet03] Java Servlet Technology
http://java.sun.com/products/servlet/; 2003

[SRVT03] Servlets Technology Overview
http://java.sun.com/products/servlet/overview.html; 2003

[STRT03] The Apache Struts Web Application Development Framework
http://jakarta.apache.org/struts/; 2003

[SunBlue03] Model view controller-java.sun.com
http://java.sun.com/blueprints/patterns/MVC-detailed.html;2003

[TmCT03] Apache Tomcat Servlet Container
http://jakarta.apache.org/tomcat/; 2003

[W3CDoM] Document Object Model
http://www.w3.org/DOM/; 2003

[W3CHTTP] Hypertext Transfer Protocol
http://www.w3.org/Protocols/; 2003

[W3CURI] Web Naming and Addressing, URIs, URLs
http://www.w3.org/Addressing/; 2003

[W3CXML] Extensible Markup Language
http://www.w3.org/XML/; 2003

[W3CXSLT] XSL Transformations
http://www.w3.org/TR/xslt ;2003

[WAC03] Web Application Controller Reference
http://www.orbeon.com/oxf/doc/reference-controller; 2003

[XForms03] XForms - The Next Generation of Web Forms
http://www.w3.org/MarkUp/Forms/; 2003

[XSLT03] XSL Transformations
http://www.w3.org/TR/xslt; 2003