A Software Architecture for Structuring Complex Web Applications

Mark D. Jacyntho, Daniel Schwabe
Dept. of Informatics, PUC-Rio, Brazil
{mark,schwabe}@inf.puc-rio.br

Gustavo Rossi
LIFIA, Univ. de La Plata, Argentina
gustavo@sol.info.unlp.edu.ar

Outline

- Motivation
- Overview of the Approach
- Proposed Framework
- Discussion
- Future Work

Motivation

- Increasing complexity of Web-based applications
  - Read only → full functionality
- Increased speed of development
- Increased speed of the feedback loop
  - Constant revision/evolution
- Need for greater productivity

Goals

- To provide a software architecture to support web-based applications of varying degrees of complexity
  - Support navigation
  - Support application functionality (business logic)
- To specify an implementation framework based on this architecture
  - Supporting design reuse
  - Supporting code reuse
- Develop a set of guidelines and design patterns to help using the framework
- Implement the framework using current application server architecture

Underlying Ideas - OOHDM Principles

- Applications are part of *man-machine team* that together solves the problem
  - Hypermedia for integration with computer-processed knowledge
  - Hypermedia to support humans
- User navigates in *Nodes*, which are *views* over Conceptual Objects
- Nodes are organized into *Navigational Contexts* – sets of objects relevant to tasks
- There is a clear distinction between Interface operations and Navigation Operations
- Interface can be specified at an Abstract Level

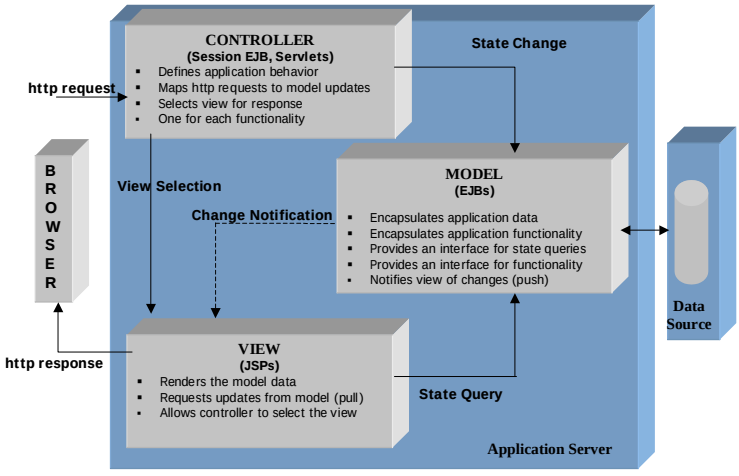
Underlying Ideas – J2EE, MVC

- Use Java 2 Enterprise Edition (J2EE)
  - Widely used in industry
  - Provides several system services, such as security, concurrency control, transactions, etc...
  - Many suppliers on the server side
  - Allows component-based development
  - Many possible types of clients
  - Allows configuration through XML files
- Use well-known architectural approaches such as the Model-View-Controller

Model-View-Controller (MVC) Architecture

- Clear separation between functionality logic, data and presentation logic;
- *Model* – represents application data (OOHDM Conceptual Model);
- *View* – Data presentation to the client.
- *Controller* – defines application behavior.
  - Translates user actions into events to be processed over the *model*
  - Selects a *view* to present as a response to the client

The MVC Architecture



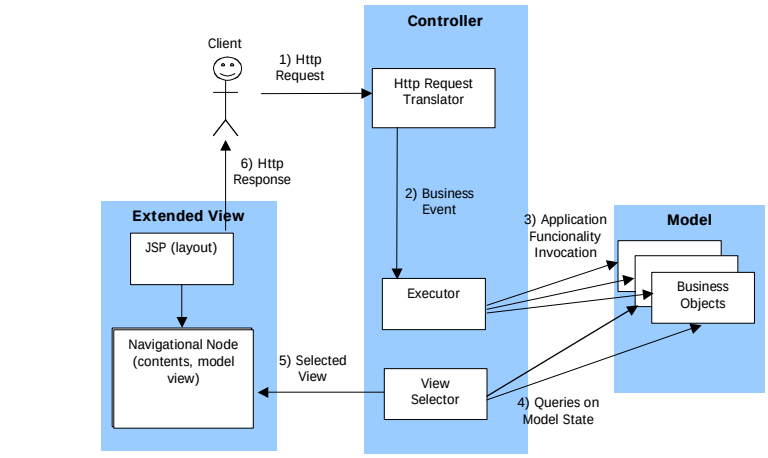
The OOHDM-Java2 Framework

- Based on the MVC architecture;
- Defines an architecture for implementing web applications.
- Allows separation of concerns: web designers, programmers;
- Eases maintenance and reuse;
- Covers both navigation (read-only) and fully functional applications;
- Based on the J2EE platform.
- Provides direct support for applications designed using the OOHDM approach.
  - May be used for applications designed using other methods

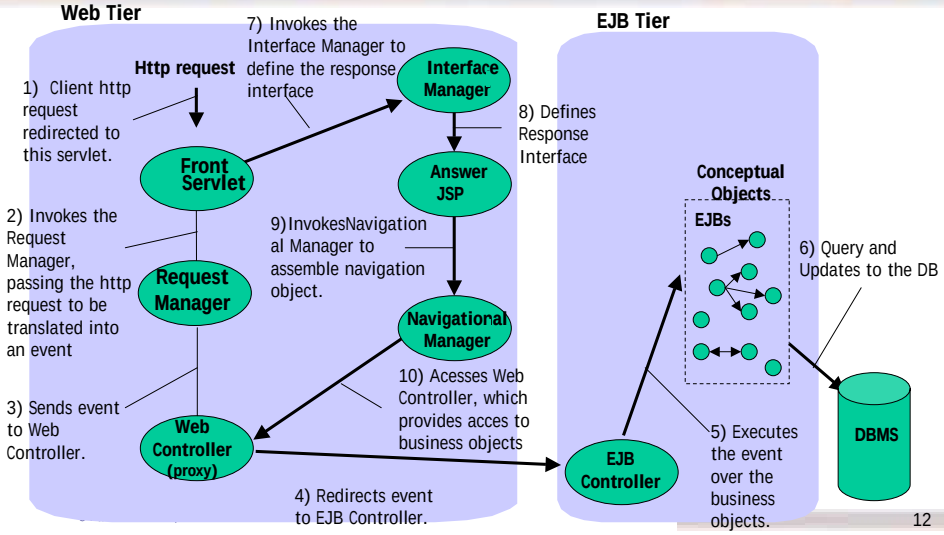
OOHDM-Java2 Modules

- Transactional
  - Implements application functionality (business logic);
  - Supports the execution of an event over the *model*;
- Navigational
  - Implements navigation operations as specified using OOHDM
  - Provides the instantiation of nodes in the corresponding context, or of the access structure being navigated
  - Provides exhibition of the appropriate navigation element

Architecture Overview



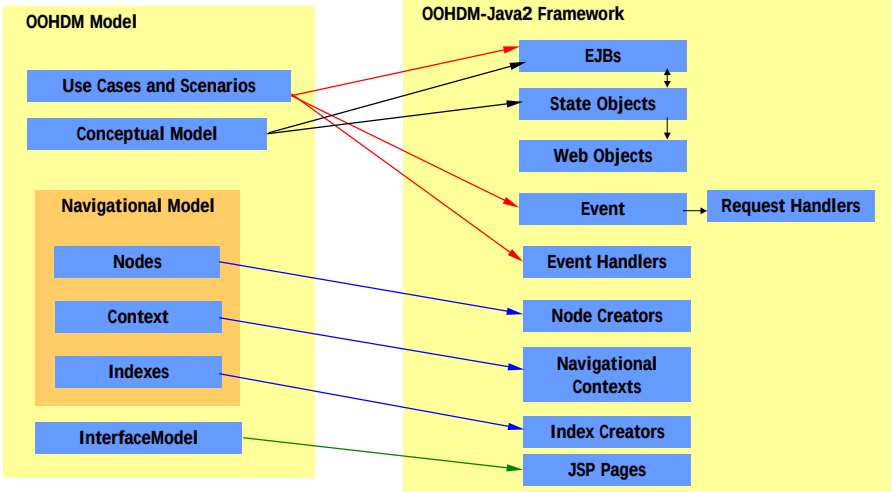
More Detailed view



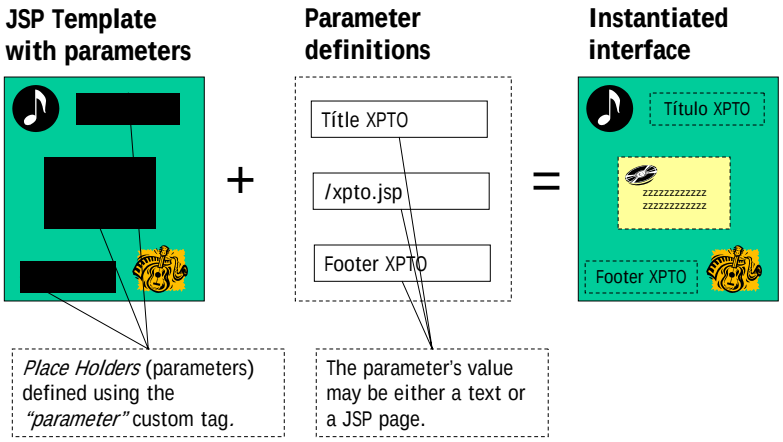
Instantiating OOHDM-Java2

1. Define the structure and behavior of application business objects
2. Define the business events in the application
3. Customize the Executor component by indicating the execution logic for each business event object.
4. Specialize the View Selector component adding the application's specific logic to select the response interface.
5. Identify the meaningful contexts (sets) of nodes, specializing the Navigational Context component.
6. Define the structure of nodes in the application, by refining the Navigational Node component.
7. Define the layout for the corresponding navigational node structure by specifying the JSP pages in the application.

Mapping between OOHDM and OOHDM-Java2



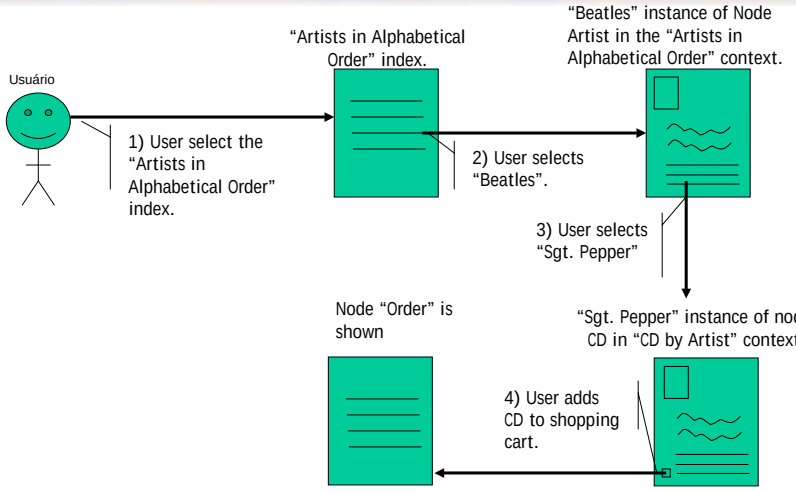
Interface Definition



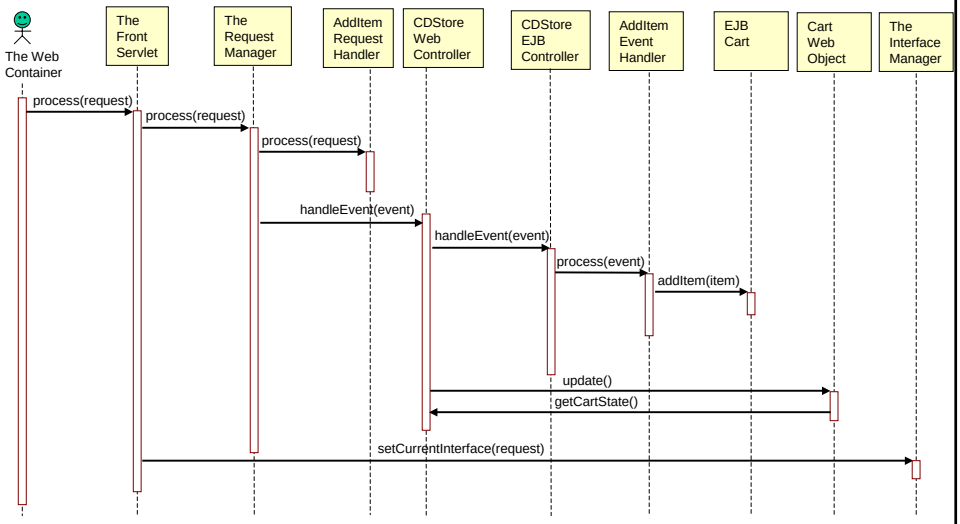
Template.jsp – CD Store

```
<%@page contentType="text/html"%>
<%@ taglib uri="/WEB-INF/taglib.tld" prefix="oohdmjava2" %>
<html>
        <head>
                <title>
                        <oohdmjava2:parameter name="HmtlTitle"/>
                </title>
        </head>
        <body>
                <table height="85%" width="100%"  cellspacing="0" border="0">
                <tr>
                        <td valign="top">
                        <oohdmjava2:parameter name="HtmlBody"/>
                        </td>
                </tr>
                <tr>
                        <td valign="bottom">
                        <oohdmjava2:parameter name="HtmlFooter"/>
                        </td>
                </tr>
                </table>
        <a href="/lojdcd/index.html"> <font size="5">CD Store Home Page</font> </a>
        </body>
</html>
```

A CD Store Scenario



Transactional Model – *"AddItemEvent"*



Transactional Model Configuration XMLs

- urlmappings.xml – Maps the request's URL to its response interface and, if necessary, to the associated request handler or interface handler.

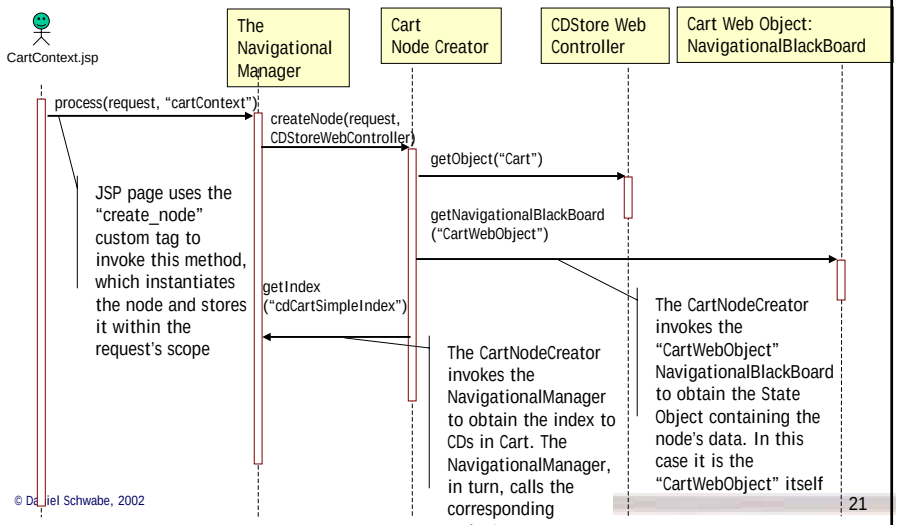
```
. . .
<url_mapping path = "/additemcart" interface = "CART_CONTEXT">
     <request_handler
       class="pucrio.inf.oohdmjava2.cdstore.web.requesthandlers.cart.AddItemRequestHandler"/>
</url_mapping>
<url_mapping path = "/removeitemcart" interface = "CART_CONTEXT">
     <request_handler
       class = "pucrio.inf.oohdmjava2.cdstore.web.requesthandlers.cart.RemoveItemRequestHandler"/>
</url_mapping>
. . .
```


Transactional Model Configuration XMLs

- eventmappings.xml – Maps each event to its corresponding event handler;
- exceptionmappings.xml – Maps each event exception to its corresponding exception handler and/or response
- interface interfaces.xml – For each interface, defines its exhibition template and its parameter values

```
. . .
<interface name="Order_Per_Client_CONTEXT" template="/template.jsp">
    <parameter name="HtmlTitle" value="Client Order." as_is="yes"/>
    <parameter name="HtmlBody" value="/OrderByClientContext.jsp" as_is="no"/>
    <parameter name="HtmlFooter" value="OOHDM-JAVA2 CD Store - Context: Client Orders."
      as_is="yes"/>
</interface>
<interface name="Cart_CONTEXT" template="/template.jsp">
   <parameter name="HtmlTitle" value="Shopping Cart." as_is="yes"/>
   <parameter name="HtmlBody" value="/CartContext.jsp" as_is="no"/>
   <parameter name="HtmlFooter" value="OOHDM-JAVA2 CD Store - Context: Shopping Cart."
     as_is="yes"/>
</interface>
. . .
```

Navigational Module – Creation of the "Cart" node



Navigational Module Configuration XML

- **contextmappings.xml – Defines the application's navigational contexts. For each one, it defines:**
  - **Context ID;**
  - **Internal navigation type;**
  - ***Navigational Context* (the Java class name for the context)**
  - **If it is a homogeneous context, a *Node Creator* and an URL;**
  - **If it is heterogeneous, a *Node Creator* and an URL for each navigational class.**

```
. . .
<context_mapping context_id = "ArtistByCDContext"
                 navigation_type = "SI"
                 context_class =
"pucrio.inf.oohdmjava2.CDStore.web.navigational.cd.contexts.byArtist.CDByArtistContext"/>
                <simple url_path = "/cdbyartistcontext"
                node_creator_class =
"pucrio.inf.oohdmjava2. CDStore.web.navigational.cd.contexts. byArtist.CDByArtistNodeCreator"/>
</context_mapping>
. . .
```

Navigational Module Configuration XML

- Indexmappings.xml – Defines the indexes in the applicatoin. Indexes with a corresponing *Index Creator* are defined first, followed by index groups and hierarchical indexes. For each one, it defines:
  - Index ID
  - *Index Creator* (if it exists);
  - URL (if it exists).

```
. . .
<index_mapping index_id = “genreAlfaSimpleIndex“
                    index_creator_class  =
"pucrio.inf.oohdmjava2.CDStore.web.navigational.cd.indexes.GenreAlphaIndexCreator"/>
<index_mapping index_id = “CDbyGenreSimpleIndex“
                    index_creator_class =
"pucrio.inf.oohdmjava2.CDStore.web.navigational.cd.indexes.CDByGenreIndexCreator"/>
<hierarch_index_mapping index_id = "genreCDHierarchIndex"
            "url_path= "/genrecdhierarchindex">
          <index_ref index_id = "genreAlfaSimpleIndex"/>        <!– first level -->
          <index_ref index_id = “CDByGenreSimpleIndex"/>        <!– second level -->
</hierarch_index_mapping>
. . .
```


Summary of OOHDM-Java2

- Support for a clear separation between application and presentation logic
- Further separation (wrt MVC) between navigation logic and interface aspects
- Support for navigational contexts and set-based navigation
- Decoupling between JSP pages and business events
- Centralized control of http requests
  - translation of http requests into business events
- Centralized control of business events execution
- Centralized selection of response interfaces
- Single entry points (Façades) to business objects, both in the Web and EJB layers.
- Single entry point for serializing requests of the same user
- Centralized mapping of business events into corresponding execution logic
- Centralized control of navigation logic

Future Work

- Definition of domain-dependent frameworks
- Definition of custom tags for each Navigational Component (e.g., Simple Indes, Anchor, Attribute, etc...)
- Automatic translation of OOHDM-ML specifications into code skeletons
- Re-instantiating the architecture in other implementation environments, e.g., .NET

Thank you!

Further material available at

http://www.telemidia.puc-rio.br/oohdm/oohdm.html