OOHDMDA – An MDA Approach for OOHDM

Hans Albrecht Schmid and Oliver Donnerhak

University of Applied Sciences Konstanz, Brauneggerstr. 55, D 78462 Konstanz
schmidha@fh-konstanz.de

Abstract. The MDA approach "OOHDMDA" generates servlet-based Web applications from OOHDM. An OOHDM application model, built with a UML design tool, is complemented with the recently proposed behavioral OOHDM semantics to serve as a PIM. This paper describes the transformation from a PIM XMI-file into a PSM XMI-file for a servlet-based PSM. It is performed by an XMINavigationalTransformer, which contains an XMI parser and a transformation class for each transformation rule.

1 Introduction and Related Work

The model-driven architecture (MDA) [1] models the business aspects of an application in a platform-independent model (PIM). The technological aspects are added when the PIM is transformed into a platform-specific model (PSM).

The modeling and design method OOHDM [2] describes hypermedia-based Web applications by an object model on three levels: the conceptual level, the navigational level, and the interface level. OOHDM is well-suited as a starting point for MDA since it is a platform-independent model, and it is an object model, so that the object classes may be easily transformed. But it has no well-defined semantics.

Therefore, we use an OOHDM application model only as a base PIM, adding to it the behavioral semantics definition of OOHDM core features and business processes [3]. This semantics derives application-related OOHDM classes from behavioral model classes with a predefined semantics, which is well-defined and executable.

This paper describes the OOHDMDA approach for a servlet-based PSM (see [4] for an overview). It generates from an OOHDM application model and the behavioral semantics model a servlet-based Web application front-end that accesses backend classes. For lack of space, we must restrict the paper to present the transformation of dynamic navigation (which is sufficiently complex) as only example, though OOHDMDA covers currently all core constructs of OOHDM (but no contexts, etc.) together with business processes [5].

Different Web application design methods, like WebML [6], UWE [7], OO-H [7], and OOWS [8], generate code from the Web page design or a design model. OOWS captures functional system requirements formally to construct from them the Web application. [9] compares the annotation approach and the diagram approach for the automatic construction of Web applications.

After an overview on the MDA-process with OOHDM in section 2, we describe the PIM for dynamic navigation with the behavioral semantics definition in section 3. Section 4 presents for dynamic navigation the transformation to a servlet-based PSM as platform-specific model.

2 MDA Process

A Web application designer designs with a UML-based design tool the OOHDM conceptual and navigational schema of a Web application (without behavioral model classes) as the **Base PIM** for the MDA process (see Fig. 1.). The OOHDM classes are to be marked with a stereotype indicating the model class, from which the OOHDM class is derived

The **Base PIM to PIM transformation** transforms the output XMI-file of the design tool to a modified UML class diagram: it replaces navigational links by model classes; it derives the base PIM classes from the model classes according to the stereotype and adds the model classes; it adds directed associations from nodes to the associated conceptual schema entities; and does further smaller transformations.

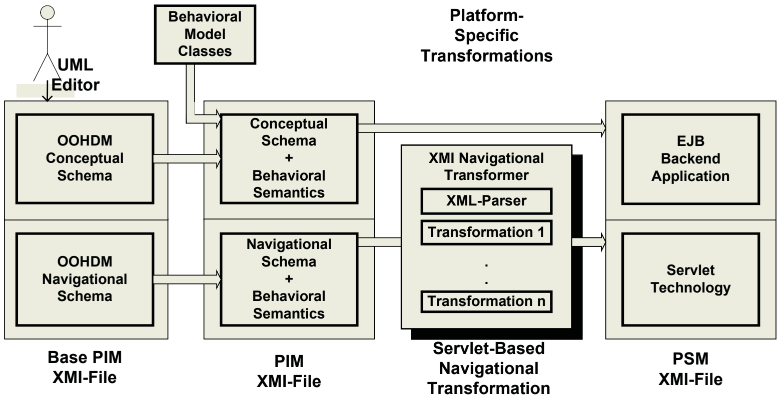

UML Editor
Behavioral Model Classes
Platform-Specific Transformations
OOHDM Conceptual Schema
OOHDM Navigational Schema
Conceptual Schema + Behavioral Semantics
Navigational Schema + Behavioral Semantics
XMI Navigational Transformer
XML-Parser
Transformation 1
Transformation n
EJB Backend Application
Servlet Technology
Base PIM XMI-File
PIM XMI-File
Servlet-Based Navigational Transformation
PSM XMI-File

Fig. 1. Conceptual and navigational Base PIM, PIM and transformation to servlet-based Navigational PSM with XMINavigationalTransformer

The OOHDM conceptual and navigational schemas represent two different, relatively independent aspects of a Web application, the Web front-end, and the application backend. Consequently, we partition also the PIM into a **Conceptual PIM** sub-model and the **Navigational PIM** sub-model (see Fig. 1.). The Conceptual Transformation and Navigational Transformation (see Fig. 1.) are completely independent, except for the operation invocations of Conceptual PSM objects from the Navigational PSM, where the kind of invocation may vary. Thus, you may select and combine the implementation technology and platform of the Conceptual PSM and the Navigational PSM quite independently, as [4] shows.

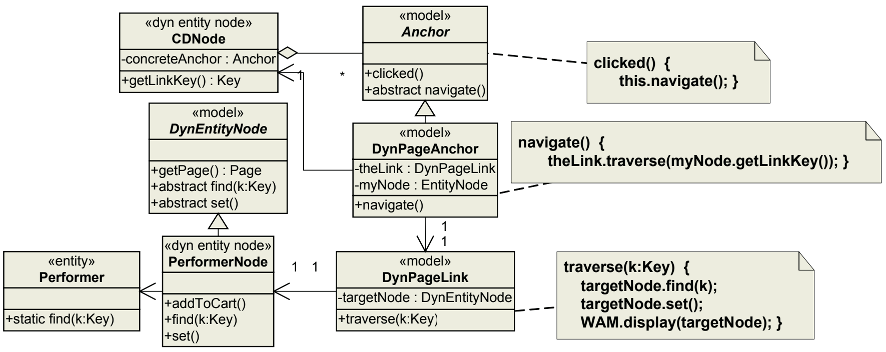
This paper focuses on the **Navigational Transformation** from the Navigational PIM into a servlet-based Navigational PSM, both represented by files in XMI format. It is described by transformation rules. Since we could not find a transformation tool to be parameterized with the transformation rules meeting our requirements, we developed an XMINavigationalTransformer, which contains an XMI parser, and for each transformation rule a transformation class. It generates as output an XMI file, from which the PSM to code transformation (not shown in Fig. 1.) generates Java code.

3 Navigational PIM for Dynamic Navigation

The OOHDM behavioral semantics derives the OOHDM application model from behavioral model classes, as e.g. conceptual schema entities, like CD, from a model class, like Entity or subclasses, and navigational schema nodes, like CDNode, from a model class, like Node or subclasses. Model classes collaborate with a Web Application virtual Machine (WAM), which models basic Web-browser characteristics, i.e. HTTP-HTML characteristics, as seen from a Web application. Both model classes and WAM have a well-defined behavioral semantics [3].

Class Node defines the operations: getPage(): Page, getField(n:Name): Value, setField(n: Name, v: Value), getFieldNames(): Name [], which are mainly used by the WAM to display the content of a page. A Node refers to the entity or entities it displays, and contains an array of InteractionElements like Anchor's or Button's, and a Page. Node has subclasses FixedEntityNode and DynEntityNode that represent pages with a fixed content and dynamically generated content.

We distinguish two kinds of navigation, navigation to a Web page with fixed content and dynamic content, i.e. navigation to a FixedEntityNode and DynEntityNode. We present the PIM for the latter one.



```
clicked() {
    this.navigate(); }
```

```
navigate() {
    theLink.traverse(myNode.getLinkKey()); }
```

```
traverse(k:Key) {
   targetNode.find(k);
   targetNode.set();
   WAM.display(targetNode); }
```

Fig. 2. PIM for dynamic navigation from CDNode to PerformerNode

Fig. 2. shows the PIM for a dynamic link in the navigational schema, like one from (the user-defined classes) CDNode to PerformerNode. The source node, like CDNode, references a DynPageAnchor that references a DynPageLink, which references the target node of the link, a DynEntityNode like PerformerNode. These references are set by a constructor parameter when the model classes are configured to work together.

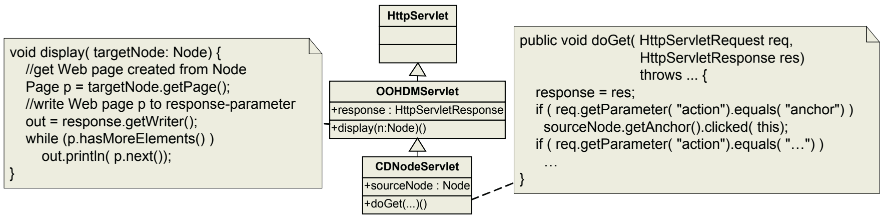
The WAM has the attribute currentNode, which references the currently displayed Node. When a user clicks at an InteractionElement of the currently displayed Web page, like the anchor of a dynamic link on the CD Web page, the WAM calls the clicked-operation of the corresponding InteractionElement of the currentNode, like that of DynPageAnchor of CDNode, which forwards the call to the navigate operation. The navigate-operation fetches the key of the dynamic content that the target node should display, from the source node CDNode (referenced by attribute myNode), calling its getLinkKey-method that returns a key, like a Performer name. Then it calls the traverse-operation, passing the key as a parameter.

The traverse-operation of DynPageLink calls the find-operation of its target node, like PerformerNode with the key as a parameter, and then its set-operation so that the target node sets its dynamically generated content. Last, traverse calls the display-operation of the WAM with the target node as a parameter. The method display(n: Node) sets that node n as the current node and calls its getPage-operation to display the page.

4 Transformation to Servlet-Based Navigational PSM

A servlet connects the backend application with the Web; it runs on a Web server, receiving an HTTP request as a parameter of a doGet- or similar operation, and sending out a HTTP response as a result of the operation. The doGet-method analyses the user input and creates the new Web page as output.

The processing performed by a servlet is similar to the processing performed by the WAM in the Navigational PIM. The doGet-method of a servlet is triggered by a user interaction and reacts on that interaction by creating a Web page as a response, in the same way, as the OOHDM behavioral model is triggered by the WAM on a user interaction and creates and displays a mask for a Web page on the WAM.



```
void display( targetNode: Node) {
   //get Web page created from Node
   Page p = targetNode.getPage();
   //write Web page p to response-parameter
   out = response.getWriter();
   while (p.hasMoreElements() )
      out.println( p.next());
}
```

```
public void doGet( HttpServletRequest req,
                   HttpServletResponse res)
                   throws ... {
   response = res;
   if ( req.getParameter( "action").equals( "anchor") )
     sourceNode.getAnchor().clicked( this);
   if ( req.getParameter( "action").equals( "...") )
     ...
}
```

Fig. 3. PSM-classes CDNodeServlet with doGet-method analyzing request parameter and calling clicked-method of the clicked-at InteractionElement, and OOHDMDAServlet

As a consequence, the navigational transformation replaces the WAM by a servlet. The **EntityNodeToServlet** transformation rule generates from each PIM Entity Node class, like CDNode, a PSM servlet class, like CDNodeServlet (see Fig. 4.), that has a reference to the node, like PSM::CDNode, which is not modified from the PIM. When a user presses an interaction element of the Web page, the doGet-method of the generated servlet analyses the response parameter and calls the clicked-method of the pressed InteractionElement of the referenced node (see Fig. 3.).

The navigational transformation modifies also the Navigational PIM classes Anchor, PageAnchor, and Link, such that the new page is not displayed by the WAM, but put into the response-parameter of the doGet-method. Doing that straightforwardly would result in the Navigational PSM being very different from the Navigational PIM, which would make the navigational transformation a complex expenditure. To keep the transformation as simple as possible, we developed the solution that the servlet provides, similarly as the WAM, a display-method which puts the node into the response parameter.

Since that responsibility is identical for all node servlets, we introduce with the **EntityNodeToServlet** transformation rule the PSM class OOHDMDAServlet, extending HttpServlet, as common superclass of all NodeServlet classes (see Fig. 3.). Its method display(targetNode: Node) gets the associated Page from the parameter targetNode; since it has no direct access to the response parameter of doGet, it writes the Page to the member variable “response” that refers to the HttpResponse, after an assignment by the doGet-method (see Fig. 3.). Thus, the page contained in the parameter targetNode is put as content into the response parameter and displayed as Web page at the return from the doGet-method call.

The navigational transformation rules **Anchor**, **PageAnchor**, and **Link** modify the clicked-method of the class Anchor, the navigate-method of the class DynPageAnchor, and traverse-method of DynPageLink so that the traverse-method of DynPageLink can call the display-method provided by the servlet: a reference to the servlet is added as an additional parameter to these methods and forwarded from call to call.

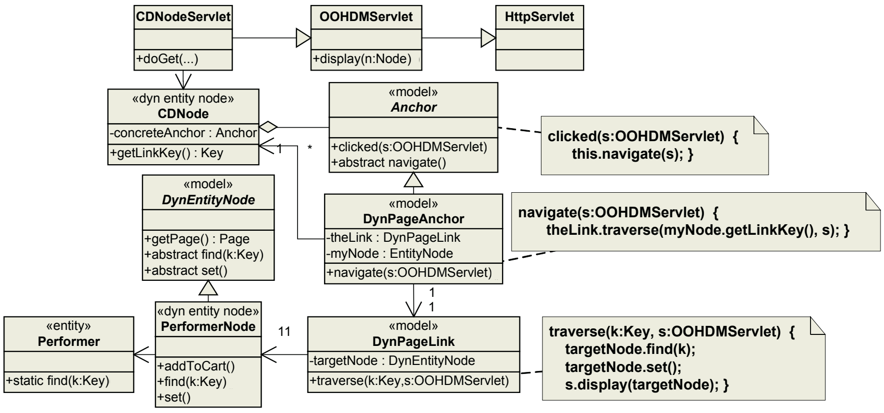


Fig. 4. Servlet-based PSM for dynamic navigation

Fig. 4. shows the resulting Navigational PSM for dynamic navigation. The method doGet of CDNodeServlet, which has a reference to CDNode, calls the clicked-method of DynPageAnchor, passing a reference to the servlet as a parameter. The transformed behavioral model classes collaborate in the same way as described in section 3, passing additionally a reference to CDNodeServlet as a parameter. The traverse-method calls the find- and set-method of the target node so that PerformerNode gets the dynamic page content from the DynEntity Performer, and inserts it into the DynPage that contains already the static HTML page content. Then, traverse calls the display-method of CDNodeServlet with PerformerNode as a parameter.

5 Conclusions

The OOHDMDA approach generates servlet-based Web applications from an OOHDM design model with the behavioral semantics as a PIM. It includes all core constructs of OOHDM and the business process extension [3] though dynamic navigation was given as only example. Based on atomic transformation rules, we have constructed an XMINavigationalTransformer, which transforms the PIM XMI-file into a PSM XMI-file. The trivial PSM-code transformation generates from a PSM

XMI-file executable Java code, which works quite efficiently. Our first experiences with the OOHDMDA approach and tool are very encouraging.

Acknowledgements

Our thanks are due to Gustavo Rossi for hosting Oliver in La plata, the International Bureau of the BMBF, Germany, for the Bilateral Cooperation with Argentina support; and the Ministerium fuer Wissenschaft und Forschung, Baden-Württemberg for a partial support of the project.

References

1. http://www.omg.org/mda/
2. D. Schwabe, G. Rossi: "An object-oriented approach to web-based application design". Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, v. 4#4, pp.207-225, October, 1998
3. H. A. Schmid, O.Herfort "A Behavioral Semantics of OOHDM Core Features and of its Business Process Extension". In Proceedings ICWE 2004, Springer LNCS, 2004
4. H. A. Schmid: "Model Driven Architecture with OOHDM". Engineering Advanced WebApplications, Proceedings of Workshops in Connection with the 4th ICWE, Munich, Germany, 2004, Rinton Press, Princeton, USA
5. H. A. Schmid, G. Rossi "Modeling and Designing Processes in E-Commerce Applications". IEEE Internet Computing, January 2004
6. S. Ceri, P. Fraternali, S. Paraboschi: "Web Modeling Language (WebML): a modeling language for designing Web sites". Procs 9th. International World Wide Web Conference, Elsevier 2000, pp 137-157
7. N.Koch, A.Kraus, C.Cachero, S.Melia: "Modeling Web Business Processes with OO-H and UWE". Procs. IWWOST 03 Workshop, Oviedo, Spain, 2003
8. O.Pastor, J.Fons, V.Pelechano: "OOWS: A Method to Develop Web Applications from Web-Oriented conceptual Models". Procs. IWWOST 03 Workshop, Oviedo, Spain, 2003
9. M. Taguchi, K. Jamroendeararasame, K. Asami and T. Tokuda "Comparison of Two Approaches for Automatic Construction of Web Applications: Annotation Approach and Diagram Approach" In Procs ICWE 2004, Springer LNCS 3140, Springer, Berlin, 2004