

# The PIM to Servlet-Based PSM Transformation with OOHDMDA

Hans Albrecht Schmid

University of Applied Sciences Konstanz,  
Brauneggerstr. 55  
D 78462 Konstanz  
xx49-07531-206-631 or -500  
schmidha@fh-konstanz.de

Oliver Donnerhak

University of Applied Sciences Konstanz,  
Brauneggerstr. 55  
D 78462 Konstanz  
xx49-07531-206-631 or -500  
o.donnerhak@kinemotion.de

## ABSTRACT

OOHDMDA generates servlet-based Web applications from OOHDMD. An OOHDMD application model built with a UML design tool is complemented with the recently proposed behavioral OOHDMD semantics to serve as a PIM. This paper describes the transformation from the PIM to a servlet-based PSM, which have a great semantic distance. Therefore, the navigational transformation applies intelligent techniques in order to avoid a very great complexity. The resulting transformation rules are quite simple. The XMINavigational-Transformer implements each rule by a transformation class that uses the services provided by an XMI parser; it transforms the PIM XMI-file into a PSM XMI-file.

## 1. INTRODUCTION

The Object-Oriented Hypermedia Design Method OOHDMD by Schwabe and Rossi [SR98] is a modeling and design method, which describes hypermedia-based Web applications by an object model on three levels: the conceptual level, the navigational level, and the interface level. OOHDMD may be considered as a platform-independent domain-specific language for Web applications that provides an object model, in contrast to other Web application modeling languages.

A domain-specific language that is to be used as a platform-independent model (PIM) for a model-driven architecture (MDA) [OMG MDA], should have a well-defined formal semantics. Therefore, we use an OOHDMD application model only as a base PIM, adding to it the behavioral semantics definition of OOHDMD core features and business processes, proposed by Schmid and Herfort [SH04]. The behavioral semantics definition derives the application-related OOHDMD classes from behavioral model classes with a fixed predefined behavioral semantics, so that they are well-defined and executable.

Schmid [S04] gives an overview over OOHDMDA, which covers all core constructs of OOHDMD together with business processes [SR04]. This paper describes in detail the transformation of the PIM to a servlet-based PSM, and in particular the transformation rules used to perform the transformation. This paper presents the transformation of all core constructs of OOHDMD, including fixed page, dynamic page and advanced navigation, which are considerably more complex than the usual MDA transformations [JCP01] [KWB04]. The transformation generates from an OOHDMD application model and the behavioral semantics model a servlet-based Web application front-end, which accesses backend classes.

After an overview on the MDA-process with OOHDMD in section 2, we describe the PIM for dynamic navigation with the behavioral semantics definition in section 3. Section 4 presents the transformation to a servlet-based PSM. Section 5 presents the transformation rules in detail. Section 6 shows how the transformations rules are implemented; section 7 presents an optimization of the transformation that reduces the number of the classes to be transformed, and section 8 presents related work

## 2. MDA PROCESS

### 2.1 Base PIM and PIM

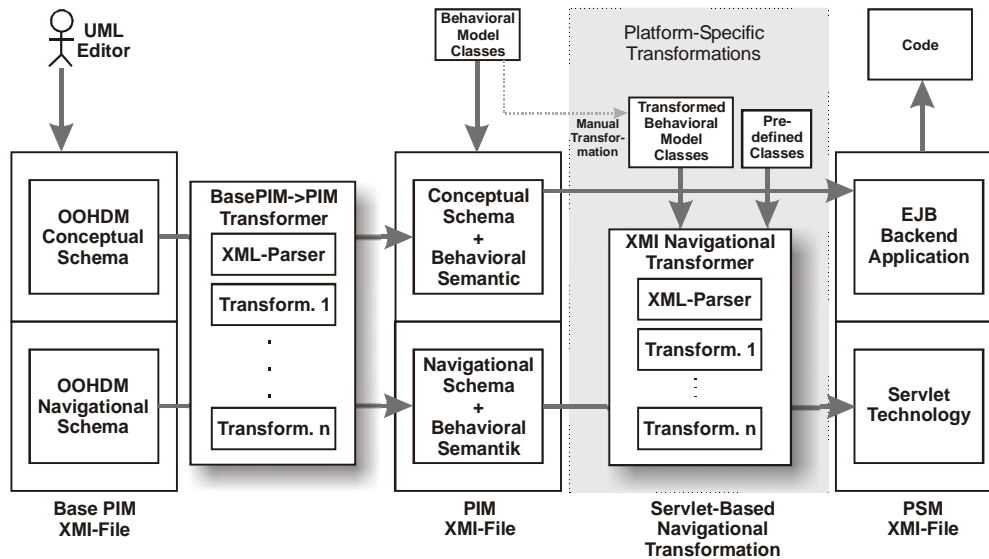
A Web application designer designs with OOHDMDA (see Figure 1) the OOHDMD conceptual and navigational schema of a Web application as the base PIM for the MDA process, using any UML-based design tool, like Rational Rose or Poseidon, that produces an XMI-file as output. The designer has to mark the application-related OOHDMD classes with a stereotype indicating the model class, from which the OOHDMD class is derived (see [SH04]).

The **Base PIM to PIM transformation** transforms the output XMI-file of the design tool to a XMI-file with a modified UML class diagram. The main transformation rules are: replace navigational links by model classes for navigational links; derive the base PIM classes from model classes according to the stereotype; add the model classes to the PIM; add directed associations from nodes to the associated conceptual schema entities.

The OOHDMD 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 Figure 1).

### 2.2 Navigational Transformation

The conceptual and navigational PIM to PSM transformation are completely independent, except for 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 [S04] shows.



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

For example, you may transform the conceptual PIM into an EJB-based conceptual PSM in order to implement the application backend with Enterprise JavaBeans (EJB). This conceptual transformation, which is not very complex, transforms the different categories of objects from the conceptual schema into different kinds of Enterprise JavaBeans, as described shortly in [S04a].

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, we constructed an XMINavigationalTransformer that implements each rule by a transformation class that uses the services provided by an XMI parser; it transforms the PIM XMI-file into a PSM XMI-file.

The simple PSM-code transformation generates from a PSM XMI-file executable Java code, that is Java servlets and classes, which work quite efficiently.

### 3. PIM MODEL CONSTRUCTS FOR FIXED, DYNAMIC AND ADVANCED NAVIGATION

The OOHDH behavioral semantics derives the OOHDH application model from behavioral model classes. That means conceptual schema entities, like CD, are derived 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 [SH04].

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. Advanced navigation allows a user to trigger an action that enters or updates information in entities from the conceptual schema.

We present two examples: the PIM model construct for dynamic navigation from CDNode to PerformerNode; and the PIM model construct for advanced navigation when a user triggers the execution of the addToCart-method of CDNode. For an easier understanding, we present both PIM model constructs independently from each other; that means we assume that CDNode allows either dynamic navigation or advanced navigation. When CDNode allows to do both dynamic navigation and advanced navigation, both PIM model constructs are superimposed such that there is only a single class CDNode with the union of the class members.

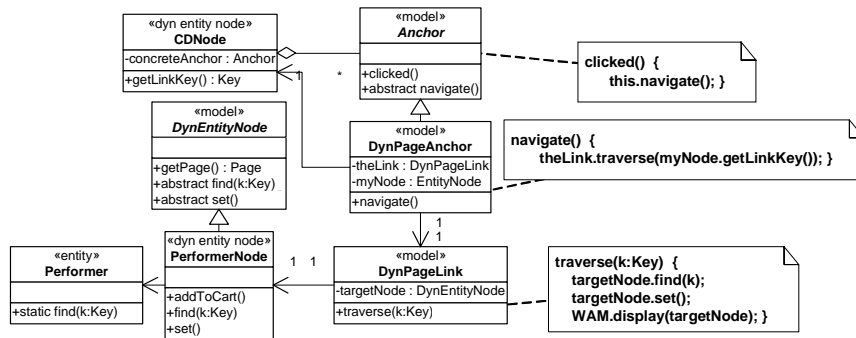


Fig. 2. PIM for dynamic navigation from CDNode to PerformerNode

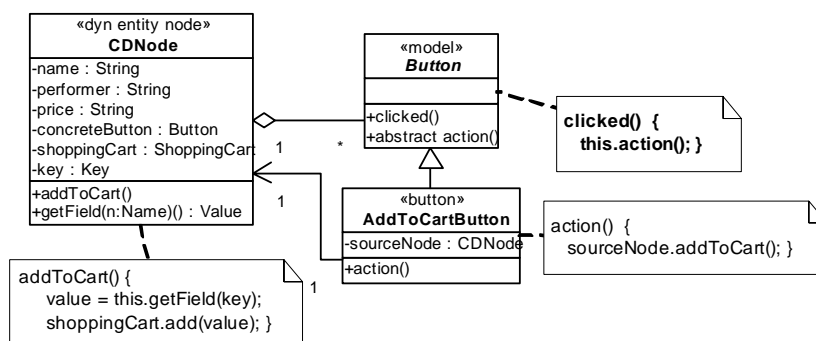


Fig. 3. PIM for advanced navigation: executing the addToCart-action triggered by AddToCartButton

### 3.1 Dynamic Page Navigation

Figure 2 shows the PIM model construct for dynamic navigation over a link from (the user-defined classes) CDNode to PerformerNode. The source node of the link, 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 constructor parameters 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.

### 3.2 Fixed Page Navigation

Fixed page navigation is a simplified dynamic page navigation. The main difference is that no key is required to identify the content of the page, since the content is always the same. Therefore, the PIM for fixed page navigation is similar to the PIM for dynamic page navigation. The differences are that the class FixedPageLink (that replaces DynPageLink) has a *traverse*-method without a key-parameter, which calls directly the *display*-method of the WAM (without the *find*- and *set*-calls of the target node); and that the *navigate*-method of class FixedPageAnchor (that replaces DynPageAnchor) does not pass a key parameter with the call of *traverse*.

### 3.3 Advanced Navigation

Figure 3 presents the PIM model construct for advanced navigation (for details, see [SH04]). Advanced navigation allows a user to trigger an atomic action by pressing a button on a Web page. An atomic action enters or edits information in a Web application, modifying the state of application objects that are modeled in the conceptual schema.

For example, consider the *addToCart* operation of the CDNode in Figure 3, which is triggered by the

AddToCartButton. The navigational PIM (see Figure 3) shows the model class Button with the operations clicked and action. A derived application-specific class, like AddToCartButton, implements the action-method, which calls an operation of the source node, like addToCart of CDNNode.

When the WAM displays a Web page, i.e. a node, and a user clicks at a button on this page, the WAM calls the clicked-operation of the corresponding InteractionElement of the currentNode, like Button. The clicked-method forwards the call to the action-method, which forwards the call to the addToCart-method of the CDNNode. This method fetches the value from the key-field of the (currently presented) CD and sends the message add(value) to the ShoppingCart object, which changes the state of the shopping cart. Note that the execution of an atomic action does not imply the navigation to another node.

#### 4. SERVLET-BASED NAVIGATIONAL PSM

This section gives an overview on the servlet-based PSM for fixed page, dynamic page and advanced navigation. The semantic distance between PIM and PSM is very great, in contrast to what is usual with MDA. Therefore, the navigational transformation applies intelligent techniques in order to avoid a very great complexity. The resulting transformation rules are quite simple. Details on them are given in section 5.

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.

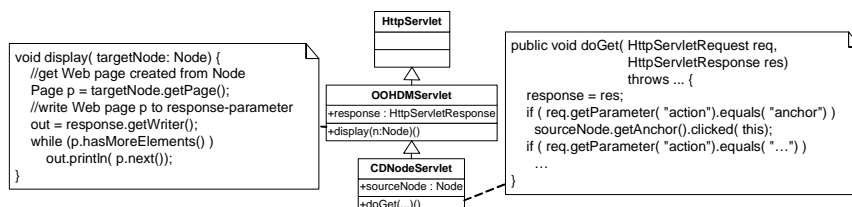
As a consequence, the **navigational transformation** replaces the WAM by a servlet. The **EntityNodeToServlet** transformation rule, which applies to all entity nodes, generates from each PIM Entity Node class, like CDNNode, a

PSM servlet class, like CDNNodeServlet (see Figure 5), that has a reference to the node, like PSM::CDNNode, 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 Figure 4).

The **navigational transformation** modifies also the navigational PIM classes like 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 putting 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 OOHDMDServlet, extending HttpServlet, as common superclass of all NodeServlet classes.(see Figure 4). 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 HttpServletResponse, after an assignment by the doGet-method (see Figure 4). 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 **InteractionElement**, **Anchor**, **Button**, **PageAnchor**, **Link**, **Button** and **ActionUserButton** modify each the clicked-method of the class InteractionElement, Anchor or Button, the navigate-method of the class FixedPageAnchor or DynPageAnchor, the traverse-method of FixedPageLink or DynPageLink, and the action-method of UserButton, so that the traverse-method of FixedPageLink or DynPageLink or the action-method of UserButton 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.** PSM-classes CDNNodeServlet with doGet-method analyzing request parameter and calling clicked-method of the clicked-at InteractionElement, and OOHDMDServlet

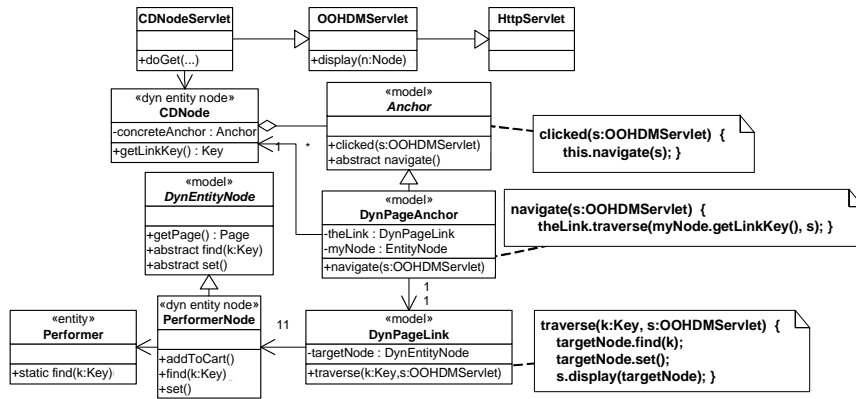


Fig. 5 Servlet-based PSM for dynamic navigation

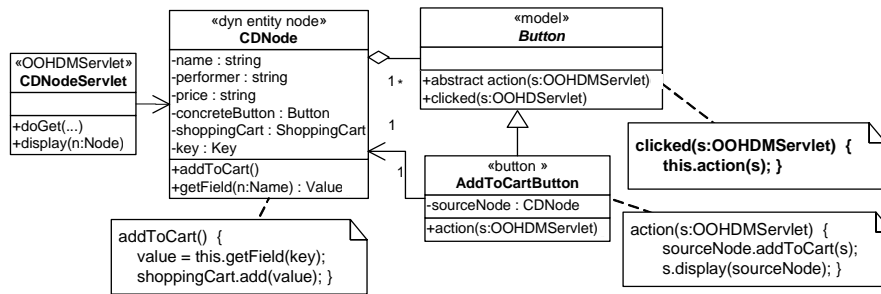


Fig. 6. Servlet-based PSM for advanced navigation

#### 4.1 Servlet-Based PSM for Dynamic Navigation

Figure 5 shows the PSM construct for dynamic navigation that is the result of the transformation. The method `doGet` of `CDNodeServlet`, which has a reference to `CDNode`, calls the `clicked`-method of the corresponding interaction element of `CDNode`, which is a `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.

The navigational PSM for fixed navigation is similar to the navigational PSM for dynamic navigation, except for the differences between the respective PIMs described in section 3.

#### 4.2 Servlet-Based PSM for Advanced Navigation

Figure 6 shows the PSM construct for advanced navigation that is the result of the transformation. The method `doGet` of `CDNodeServlet`, which has a reference to `CDNode`, calls the `clicked`-method of the corresponding interaction element of `CDNode`, which is a `Button`, passing a reference to the servlet as a

parameter. The `clicked`-method calls the `action`-method of `AddToCartButton`, passing a reference to the servlet as a parameter, which calls in turn the `addToCart`-method of `CDNode`. This method gets the key that identifies the CD presented to the user, and calls the `add`-method of the `ShoppingCart` object.

### 5. TRANSFORMATION RULES FOR THE NAVIGATIONAL TRANSFORMATION

The navigational transformation from the PIM to the servlet-based PSM is formed by a set of independent atomic transformations. We describe a transformation in this section for a better understanding by a semi-formalized transformation rule. A transformation rule has

1. a name
2. a source, which describes the PIM class or classes to be transformed. It indicates the class name and/or the stereotype of the class to be transformed, like `<<DynEntityNode>>`, and possibly additional selection criteria
3. a target, which describes the transformed PSM class or classes
4. a `ToDo` section that describes the modifications that are to be applied to the PIM class or classes to generate the PSM class or classes
5. a graph section. It represents on the left the source graph with the PIM classes and associations among them, and on the right the destination graph with the PSM classes and associations among them.

We present all transformation rules that the navigational transformation uses to transform the PIM for fixed page, dynamic page and advanced navigation. The same rules apply for fixed page and dynamic page navigation, as Table 1 shows, since the differences among them are expressed by different OOHDMD and behavioral model classes, and not generated by the MDA transformations. Advanced navigation shares two rules with them and adds two more rules. Table 1 gives an overview on the transformation rules and for which kind of navigation they are used.

Area of Use	FixedPage	DynPage	Advanced
Rule	Navigation	Navigation	Navigation
EntityNode ToServlet	X	X	X
InterAction Element	X	X	X
Anchor	X	X	
PageAnchor	X	X	
Link	X	X	
Button			X
Action			X
UserButton			

**Table 1** Overview on the transformation rules and their use

The transformation rules are presented in an abbreviated form in Figures 7 and 8.

The transformation rule **EntityNodeToServlet** (see Figure 7) is applied to all nodes derived from FixedEntityNode or DynEntityNode. It generates from a class PIM::<AnyClass>

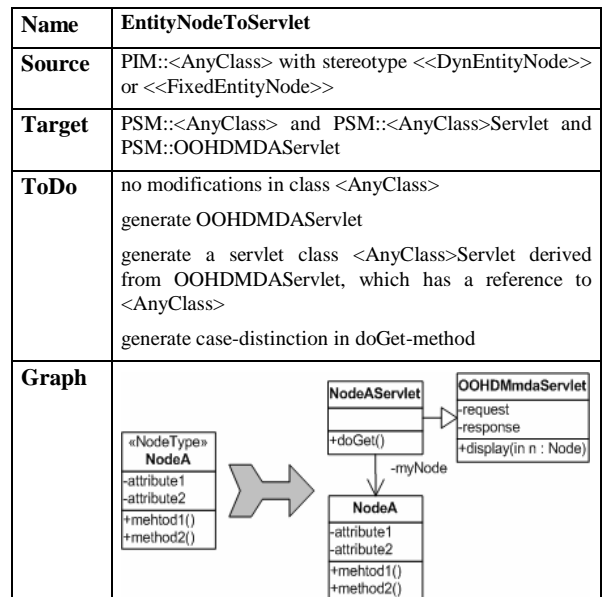
- an unmodified class PSM::<AnyClass>
- a class PSM::OOHMDAServlet (that is the same for all entity nodes)
- and a class PSM::<AnyClass>Servlet for which a doGet-method is generated as described in section 4 (see Figure 4): it analyses the request parameter to determine the anchor or button etc. clicked-at, and invokes the corresponding clicked-method of the associated node.

The transformation rule InteractionElement (see Figure 8 a) is applied to the abstract base class InteractionElement from which Anchor, Button, etc. are derived. Note that the class InteractionElement was left off in the class diagrams.

The transformation rule Anchor is applied to the class PIM::Anchor, PageAnchor is applied to PIM::FixedPageAnchor or PIM::DynPageAnchor, Link is applied to PIM::FixedPageLink or PIM::DynPageLink (see Figure 8 b-d).

The transformation rules Anchor and PageAnchor add each a parameter of type OOHMDAServlet to the methods: clicked of Anchor, respectively navigate of FixedPageAnchor or DynPageAnchor, and modify the behavioral semantics of these methods so that each forwards the newly added parameter to the navigate-, respectively traverse-method. The transformation rule Link adds a parameter of type OOHMDAServlet to the traverse-

method, and replaces the WAM as the receiver of the display-call by the servlet parameter.



**Figure 7** Navigational transformation rule **EntityNodeToServlet**

There is only one transformation rule that is applied to transform both the navigate-methods of the classes FixedPageAnchor and DynPageAnchor, though both methods have a different number of parameters and a different behavioral semantics. The same holds for the traverse-method of the classes FixedPageLink and DynPageLink.

<b>Name</b>	<b>InteractionElement</b>
<b>Source</b>	PIM::InteractionElement
<b>Target</b>	PSM::InteractionElement
<b>ToDo</b>	add a parameter of type OOHMDAServlet to method clicked()
<b>Graph</b>	not shown, since trivial

**Figure 8 a** Navigational transformation rule InteractionElement

<b>Name</b>	<b>Anchor</b>
<b>Source</b>	PIM::Anchor
<b>Target</b>	PSM::Anchor
<b>ToDo</b>	add a parameter of type OOHMDAServlet to method clicked(); modify behavioral semantics of clicked to forward the added parameter with call of navigate-method
<b>Graph</b>	not shown, since trivial

**Figure 8 b** Navigational transformation rule Anchor

<b>Name</b>	<b>PageAnchor</b>
<b>Source</b>	PIM::FixedPageAnchor or PIM::DynPageAnchor
<b>Target</b>	PSM::FixedPageAnchor or PSM::DynPageAnchor
<b>ToDo</b>	add a parameter of type OOHMDAServlet to method navigate; modify behavioral semantics of navigate to forward the added parameter
<b>Graph</b>	not shown, since trivial

Figure 8 c Navigational transformation rule PageAnchor

<b>Name</b>	<b>Link</b>
<b>Source</b>	PIM::FixedPageLink or PIM::DynPageLink
<b>Target</b>	PSM::FixedPageLink or PSM::DynPageLink
<b>ToDo</b>	add a parameter of type OOHMDAServlet to method traverse() modify behavioral semantics code of traverse to replace the WAM as receiver of the display-call by the newly introduced parameter
<b>Graph</b>	not shown, since trivial

Figure 8 d Navigational transformation rule Link

<b>Name</b>	<b>Button</b>
<b>Source</b>	PIM::Button
<b>Target</b>	PSM::Button
<b>ToDo</b>	add a parameter of type OOHMDAServlet to method clicked(); modify behavioral semantics of clicked to forward the added parameter with call of action-method
<b>Graph</b>	not shown, since trivial

Figure 8 e Navigational transformation rule Anchor

<b>Name</b>	<b>ActionUserButton</b>
<b>Source</b>	PIM:: <AnyClass> with stereotype <<Button>>
<b>Target</b>	PSM:: <AnyClass>
<b>ToDo</b>	add a parameter s of type OOHMDAServlet to method action; modify behavioral semantics code of action-method to add at the end a display-call with s as a parameter and the sourceNode as receiver
<b>Graph</b>	not shown, since trivial

Figure 8 f Navigational transformation rule UserButton

## 6. IMPLEMENTATION OF THE TRANSFORMATION RULES

The navigational transformation is performed by the XMINavigationalTransformer (see Figure 1). It has as input and

output each an XMI file. From the output file, executable Java classes are generated.

We programmed the described transformation rules in the XMINavigationalTransformer, since we could not find a transformation tool that could handle the described transformations, which are relatively complex.

The XMINavigationalTransformer contains an XML parser, and for each transformation rule a C# class implementing that rule. A transformation class uses query functions supplied by the XML parser to find a PIM class that meets the criteria described in the source part of the transformation rule. Then, a further query function is used to find the part of the class that is to be modified, like a certain method. When e.g. a parameter is to be added, a prefabricated XMI template with placeholders is used, that means modified and inserted into the XMI tree.

We selected C# as a programming language for the transformation classes since it provides a rich XML functionality. As a consequence, the transformation classes are short, simple and well structured.

Our experience is that it is much simpler to program the transformation rules than to use a tool to implement them.

## 7. TRANSFORMATION OPTIMIZATION

The PIM behavioral model classes like PIM::InteractionElement, PIM::Anchor, PIM::FixedPageAnchor, PIM::DynPageAnchor, PIM::FixedPageLink, PIM::DynPageLink and PIM::Button have a fixed predefined semantics. As a consequence, it is not necessary to transform each time with each transformation process, when we transform the OOHDM application-model classes.

Therefore, we create once pre-transformed PSM classes: PSM::InteractionElement, PSM::Anchor, PSM::FixedPageAnchor, PSM::DynPageAnchor, PSM::FixedPageLink, PSM::DynPageLink, and PSM::Button; and add them after the navigational transformation to the transformed OOHDM application-model classes. Since the class PSM::OOHMDAServlet is the same for all nodes, we do not need to generate it with each transformation process, but provide it also as a pre-transformed PSM class.

## 8. RELATED WORK

Different Web application design methods, like WebML by Ceri, Fraternali, and Paraboschi [C00], W2000 by Baresi, Garzotto, and Paolini [B00], UWE by Koch and Kraus [KKCM03], OO-H by Cacherio and Melia [KKCM03], and OOWS by Pastor, Fons and Pelechano [PFP03], generate code from the Web page design or a design model.

We distinguish a model-based process from a code-generation process. A model-based process transforms a formal PIM model into a formal PSM model. In contrast, a code-generation process does not use a formal PIM model, but an informal model, like e.g. a user design of a Web page and a verbal semantics of that design.

We classify the approach taken by WebML and W2000 as a code generation process, and the approach taken by UWE, OO-H, and OOWS together with our approach as a model-based process. UWE and OO-H use UML as a formal PIM model language, and OOWS captures functional system requirements formally to construct from them the Web application.

Taguchi, Jamroenderarasame, Asami and Tokuda distinguish and compare two code-generation approaches, the annotation

approach and the diagram approach for the automatic construction of Web applications [TJAT04].

## 9. CONCLUSIONS

We have presented how the OOHDMDA approach generates servlet-based Web applications from an OOHDM design model. OOHDMDA comprehends all core constructs of OOHDM and the business process extension [SR04]. OOHDMDA may be complemented easily with concepts currently not included, like navigational contexts, once a behavioral semantics is defined.

The base PIM to PIM transformation, which is a small effort, is done manually. The tools provided to perform the PIM to PSM transformation and the PSM to code transformation are running.

OOHDM is used together with the recently proposed behavioral semantics model as a PIM. The advantages of a model-based transformation process with a formally defined domain-specific language as a PIM are:

1. The semantics of the Web application model is well-defined by the domain-specific PIM language, and it may be extended when required. Consider e.g. dynamic navigation: if getting the key for the page content from the old page is more complicated than described by the behavioral semantics of the `DynPageAnchor` class (see Figure 2), a class that overrides the `navigate-method` may be derived from `DynPageAnchor`.
2. The software structure of the generated Web application is well-described by the transformation rules; it may be extended or modified when required. Consider e.g. the **EntityNodeToServlet** transformation rule: it states clearly that an own servlet is generated for each entity node. If that would not be desired, it is quite easy to provide an alternative **EntityNodeToServletA** transformation rule which generates one servlet class for all entity nodes of an Web application, and to provide also an alternative transformation class implementing that modified rule.

This transparency is an important advantage if a tool is used to generate real-world applications for greater companies, since there are company-internal standards to be adhered to. Extension of the behavioral semantics or modification of the transformation rules makes it possible to adhere to different standards.

## 10. 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.

## 11. REFERENCES

- [B00] L. Baresi, F. Garzotto, and P. Paolini. "From Web Sites to Web Applications: New issues for Conceptual Modeling". In *Procs. Workshop on The World Wide Web and Conceptual Modeling*, Salt Lake City (USA), October 2000.
- [C00] 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
- [JCP01] Java Community Process Document JSR26:"UML/EJB Mapping Specification"
- [KKCM03] N.Koch, A.Kraus, C.Cachero, S.Melia: "Modeling Web Business Processes with OO-H and UWE". *IWWOST 03, Proceedings 3<sup>rd</sup> International Workshop on Web-Oriented Software Technology*, Oviedo, Spain, 2003
- [KWB04] A.Kleppe, J.Warmer, W.Bast: "MDA Explained", Addison-Wesley Pearson Education, Boston, USA, 2004
- [OMG MDA] <http://www.omg.org/mda/>
- [PPF03] O.Pastor, J.Fons, V.Pelechano: "OOWS: A Method to Develop Web Applications from Web-Oriented conceptual Models". *IWWOST 03, Proceedings 3<sup>rd</sup> International Workshop on Web-Oriented Software Technology*, Oviedo, Spain, 2003
- [S04] H. A. Schmid: "Model Driven Architecture with OOHDM". *Engineering Advanced WebApplications, Proceedings of Workshops in Connection with the 4<sup>th</sup> ICWE*, Munich, Germany, 2004, Rinton Press, Princeton, USA
- [S04a] H. A. Schmid: "Model Driven Architecture with OOHDM". *Proceedings IWWOST 04*, Munich, Germany, 2004
- [SH04] H. A. Schmid, O.Herfort "A Behavioral Semantics of OOHDM Core Features and of its Business Process Extension". In *Proceedings ICWE 2004*, Springer LNCS 3140, Springer, Berlin, 2004
- [SR04] H. A. Schmid, G. Rossi " Modeling and Designing Processes in E-Commerce Applications". *IEEE Internet Computing*, January 2004
- [SR98] 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
- [TJAT04] M. Taguchi, K. Jamroenderarasame, K. Asami and T. Tokuda "Comparison of Two Approaches for Automatic Construction of Web Applications: Annotation Approach and Diagram Approach" In *Proceedings ICWE 2004*, Springer LNCS 3140, Springer, Berlin, 2004