ABSTRACT
A model-driven architecture (MDA) transforms a platform-independent model (PIM) into a platform-specific model (PSM) before it transforms the PSM into code. OOHDM together with a recently proposed business process extension allows to model hypermedia-based and advanced Web applications by an object model which is a platform-independent. But an OOHDM model cannot directly be used as PIM since it is not an executable model. A behavioral semantics definition of the OOHDM core features and business process makes an OOHDM model executable on a platform-independent Web Application virtual Machine (WAM). As a consequence, an OOHDM behavioral semantics model can serve as a PIM for a MDA. We present two different aspects of the PIM, and give two concrete examples for platform-specific models (PSM) into which the PIM is transformed.

1. INTRODUCTION
The model-driven architecture (MDA) proposed by the OMG [OMG MDA] separates different aspects of application development, mainly the business aspects and the technology aspects, from each other. Therefore, it proposes to develop first a platform-independent model (PIM) (see Figure 1) that models the business aspects without consideration of the technological aspects. A PIM is transformed into a platform-specific model (PSM) by an automatic transformation. The code is considered a another special model that is generated by another transformation from a PSM. All models are executable.

The Object-Oriented Hypermedia Design Method OOHDM 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. OOHDM gives a platform-independent object model since the models on the conceptual, navigational and interface level do not contain any platform-specific constructs. But due to the verbal semantics definition, an OOHDM model is not executable. Thus, it cannot directly be used as a PIM for MDA. A behavioral definition of the semantics of OOHDM core features and business processes proposed by Schmid and Herfort [SH04], makes the OOHDM semantics executable on a platform-independent Web Application virtual Machine (WAM). OOHDM together with the behavioral semantics definition can serve as a PIM for a MDA.

The executable OOHDM behavioral semantics PIM may be transformed in platform-specific models. Examples for transformations to two different PSMs are given.

2. WEB APPLICATION MODEL
OOHDM
This section follows [SH04] and uses the Web shop presented in [SR04] as an example for an OOHDM model that includes navigation, advanced navigation and a business process. OOHDM (for details see [SR98]) models the objects forming the application domain in a conceptual schema (see Figure 2); it models abstracted Web pages and the navigation possibilities among them in a navigational schema (see Figure 3); and the presentation aspects of Web pages in an interface schema (which we disregard). The conceptual schema is partitioned in entities (bottom) and in processes with classes and activities (top), and the navigational schema in entity nodes, among which you may navigate (left), and in activity nodes that belong to processes (right). UML stereotypes indicate the category to which each object class belongs. Note that the classes shown in Figure 2 and 3 do not give a complete application model of the example Web shop, to avoid an overloading with details. The relationship between objects in the conceptual and navigational schema, like that between an entity node CDN0Node and an entity CD, is given explicitly in the OOHDM node definition syntax [SR98], but represented in the schemas only by the correspondence of the names.

2.1 Pure Navigation
Consider, for example, the navigation possibilities in the CD store. On the left of the navigational schema (see Figure 3), you find entity nodes (which are an abstraction of Web pages) like CDN0Node or ShoppingCartNode. Links among nodes are represented as directed edges which may be labeled with the link name. There are links representing the navigation possibilities from the customer HomePageNode to the CDN0Node or to the ShoppingCartNode, from the CDN0Node to the ShoppingCartNode and back, and from one CD to other related CDs by using the "related" link between CDN0Nodes.
2.2 Advanced Navigation

More advanced hypermedia applications are not composed of read-only pages; they use Web pages as the interface for triggering different kind of actions that may change the internal state of the application. An atomic action, like adding a product to a shopping cart, calls an operation of an entity like ShoppingCart. When the user presses the “add to shopping cart” button on the Web page that displays the CDNode, called CDNode interface on the OOHDM interface layer, that button invokes, as described on the OOHDM interface layer, the addToCart operation of the selected CDNode. The CDNode sends the message add(CD) to the ShoppingCart object, which changes its state.

Figure 2 OOHDM conceptual schema of a Web shop including entities and a business process

Figure 3 OOHDM navigational schema of a Web shop including navigation, advanced navigation and a business process
2.3 Business Processes

Processes and activities are modeled in the OOHDM conceptual schema (see Figure 2 top), and activity nodes in the OOHDM navigational schema (see Figure 3 right).

Typically, a business process like CheckOut (see Figure 2 top) is composed of several activities like Login, ConfirmItems, SelectShippingAddress, etc. This is represented by an aggregation relationship in the conceptual schema. We consider a business process as a root activity that may consist itself of a set of activities. An activity is either basic, like Login etc., or composed from other activities, like CheckOut, following the composite pattern [GHJV95]. An activity collaborates with application entities, like Login with Customer. An activity provides operations like start, next, suspend and resume, that allow to execute a business process.

An activity node like LoginNode models abstractly a Web page, presenting the output of an activity and accepting its input. The OOHDM interface layer (not shown) describes which buttons, like commit or next, a node contains. Pressing a button triggers a matching method of the activity node, like next of LoginNode, which calls the matching method of the activity, like next(d:LoginData) of Login, passing the user input. An activity node is shown in the context that is created by its process. This context is represented as an ActivityNodeContainer, like CheckOutNodeContainer (Figure 3 right).

An edge labeled with a reserved label start, terminate, suspend, resume or next does not represent a navigational link, but the possibility to go from or to get to an activity node by process execution. Informally, the edge semantics is as follows (see [SR04]). The start edge from the ShoppingCartNode to the CheckOutNodeContainer represents starting the CheckOut process from navigation, the terminate edge terminating it and taking navigation up again. A next edge among activity nodes, e.g. from LoginNode to ConfirmItemsNode, represents the transition from one to the next activity, including the completion of the first and the starting of the next activity. An activity diagram (not shown) that forms part of the conceptual schema shows the possible flow of control among the activities.

A Web application may allow that a business process is suspended to do temporarily some navigation. E.g., a user may suspend checking out at the ConfirmItemsNode and navigate to the CDNode to get more information about CDs he is buying. This is represented by a suspend edge. A resume edge from an entity node to an activity node container, like the one leading from CDNode to CheckoutNodeContainer, represents returning from the temporary navigation and resuming the business process at the state it was suspended.

3. OOHDM BEHAVIORAL SEMANTICS AS A PLATFORM-INDEPENDENT MODEL

As described in [SH04], the behavioral OOHDM semantics introduces behavioral model classes (see Figure 4) that collaborate with a Web Application virtual Machine (WAM). The WAM models basic Web-browser characteristics, i.e. HTTP-HTML characteristics as seen from a Web application. The behavioral model classes and the WAM have a predefined semantics.

We distinguish behavioral model classes that define primarily the semantics of the conceptual schema, like Entity, DynEntity, RootActivity and others that define primarily the semantics of the navigational schema, like Node, Page, PageHtmlSkeleton, InteractionElement, Anchor and Button. Both sets of classes are quite independent from each other.

Therefore, we consider the behavioral OOHDM semantics as a PIM that consists of two related sub-models, a Conceptual PIM sub-model and a Navigational PIM sub-model. As we will see in section 4, we can transform the two PIM sub-models quite independently into PSMs.

For an overview and a description of the model classes see [SH04]. Here, we give more detail about the behavioral model classes and about the WAM, which play a central role for the execution of a Web application and for the transformation to a servlet-based PSM.

3.1 PIM Behavioral Model Classes and WAM

Class Node defines the (abstract) operations getPage(): WebPage, getField n:Name): Value, setField(n: Name, v: Value), getFieldName(): Name [. It contains an array of InteractionElements like Anchor’s or Button’s and a Page. Class Page contains an array of lines of type String, and methods for adding a new line at the end and for iterating through the line array. Class DynPage extends Page by a method insert(FieldName n, String s) that allows to insert dynamic content into the HTML skeleton of a page.

Class DynEntityNode, a specialized Node, has a reference to a dynamic entity of type DynEntity, besides that to a dynamic entity instance identified by the key as parameter, which sets the matching values (of the entity instance identified by the key) into the entity attributes. A DynEntityNode like CDNode has a set-field-method, which gets the attribute-values from the associated entity and inserts them into the dynamic page, calling its insert-method.

The WAM has the attribute currentNode, a reference to the currently displayed Node, and an operation display(n:Node) \{ currentNode :=n; showPage(n.getPage());\}. When a user enters or edits data on the currently displayed page, the WAM calls the setField-method of the currentNode to change the state of the Node. When a user clicks at an interaction element, the WAM calls the operation clicked of the corresponding Anchor or Button of the currentNode.
3.2 Navigational PIM

We present two aspects of the Navigational PIM in the form of the behavioral OOHDM semantics for dynamic navigation and advanced navigation (for details, see [SH04]).

PIM for navigation among Web pages with a dynamic content

Navigation allows a user to navigate from a given node to a linked node. We distinguish two kinds of navigation, navigation among Web pages with a fixed content and navigation among Web pages with a dynamical content, and present the latter one.

Navigation to a dynamic page follows a link to a node with a dynamically generated content, like the link from ShoppingCartNode to CDNode in the navigational schema Figure 2. The behavioral model shown in Figure 4 contains, besides the user-defined classes for the source and target node of the link, only the model classes: Anchor, DynEntityAnchor, and DynPageLink; that are instantiated and configured to work together. These model classes have executable definitions of the methods clicked, navigate and traverse.

A source node of a dynamic link like the ShoppingCartNode has an operation GETLinkKey that returns a key identifying the dynamic content of the target node. It contains a DynPageAnchor instance, which, in turn, references the target node of the link. Each reference is set by a constructor parameter. The class DynPageAnchor has a navigate-operation that fetches the key of the dynamical content from the source node, called myNode, and passes it as a parameter with the call of the traverse-operation.

A DynEntityNode defines an operation find(k: Key) that fetches the dynamic node content from the associated entity (calling its static find(k:Key)-operation), and a set-operation to set the (found) dynamical content into the attribute fields. The traverse-operation of the DynPageLink class calls the find-operation of its target node of type DynEntityNode 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.

When the WAM displays a Web page, i.e. a Node, and a user clicks at the Anchor of a fixed page link, the WAM calls the clicked-operation. This forwards the call to the navigate-operation of the DynPageAnchor, which calls the traverse-operation of the DynPageLink. The traverse-operation calls the display-operation of the WAM with the target node as a parameter. The WAM sets, as described in section 3, that node as the current node and calls its getPage-operation in order to display the page.

Figure 4 PIM behavioral model classes Node, InteractionElement, Link and Page
PIM for advanced navigation

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 2, which is triggered by the AddToCartButton. The behavioral model for advanced navigation (see Figure 6) 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 CDNode.

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 operation clicked of this button. The clicked-method forwards the call to the action-method, which forwards the call to the addToCart-method of the CDNode. 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. This would have to be modeled by adding a Link to CDNode and addToCart calling additionally the traverse-method of the link.

4. PLATFORM SPECIFIC MODELS

A platform-specific model shows how an application is implemented in a specific technology on a specific platform. A Web application has two different aspects for which you may select an implementation technology quite independently: the Web front-end and the application back-end. In OOHDM, this corresponds to the independent selection of the implementation technology for the navigational schema and for the conceptual schema.
That means the implementation technology and platform for the Conceptual PIM sub-model and the Navigational PIM sub-model may be selected quite independently. The transformation from the Conceptual PIM into the Conceptual PSM, called Conceptual Transformation, and the transformation from the Navigational PIM into the Navigational PSM, called Navigational Transformation, are also done quite independently.

Figure 7 shows the Conceptual PIM and the Navigational PIM and possible transformations into different Conceptual PSMs and Navigational PSMs. For example,

- you may realize the Conceptual PIM by a backend application built from Enterprise JavaBean (EJB) objects and thus transform it into a EJB-based Conceptual PSM, or similarly into a CORBA-object-based Conceptual PSM, etc. Or you may realize the Conceptual PIM by lightweight backend objects like persistent objects that are implemented on one of the different persistent object platforms offered, so you transform it into a persistence-based Conceptual PSM. Or you may have no backend objects at all, but implement a direct database access from the front-end Web application so that you transform the Conceptual PIM into a direct-DB-access-based Conceptual PSM.
- you may realize the Navigational PIM with a servlet front-end and thus transform it into a servlet-based Navigational PSM, or you may realize it with Java Server Pages (JSP) front-end and transform it into a JSP-based Navigational PSM, or with a WACoF component front-end [SD03] and transform it into a WACoF-component-based Navigational PSM, etc.

A Conceptual PSM and a Navigational PSM may be combined quite freely. The Conceptual Transformation is independent from the Navigational Transformation. Inversely, the Navigational Transformation is nearly independent from the Conceptual Transformation; only when Navigational PIM objects invoke operations of Conceptual PIM objects, the kind of invocation may vary with different Conceptual PSMs.

5. CASE STUDY 1: EJB-BASED CONCEPTUAL PSM

Let us consider an EJB-based Conceptual PSM, that means the conceptual objects are implemented as a backend application on an Enterprise JavaBean (EJB) platform. The Conceptual Transformation transforms a conceptual object into an Enterprise JavaBean. An entity with a permanent lifetime, like a CD or a Customer, is transformed in an EJB entity bean, and a business process or activity is transformed
in a stateful EJB session bean. An entity, which has only a temporary lifetime with the duration of a session, like possibly a ShoppingCart (in the case that a customer cannot re-access a shopping cart filled in a previous shopping session), is also transformed in a stateful EJB session bean.

Figure 8 shows the EJB-based Conceptual PSM for the example Web shop. The Conceptual Transformation from the Conceptual PIM is not complex. A description of the UML-EJB transformation is given by the Java Community Process document JSR26: “UML/EJB Mapping Specification” [JCP01].

6. CASE STUDY 2: SERVLET-BASED NAVIGATIONAL PSM

Let us consider a servlet-based Navigational PSM. Servlets form in the J2EE architecture the connection among the application and the Web. A servlet is a class that runs on a Web server; it receives an HTTP request as a parameter of a doGet- or similar operation, and sends out a HTTP response as a result of the operation.

A doGet-method has primarily two responsibilities, user response analysis and Web page creation. First, it analyzes the user response to determine the interaction element, like a button, that the user has pressed on the Web page. When it has determined the interaction element, it carries out the action to be done on that user interaction. Eventually, it creates a Web page as response to the user request and stores it in the response parameter of the doGet-method.

6.1 Navigational Transformation

From this description we see that the processing done by the WAM in the Navigational PIM is similar to the processing done by a servlet. In the same way as the OOHDM behavioral model is triggered by the WAM on an user interaction, and creates and displays a Web page on the WAM (compare section 3.2), the doGet-or a similar method of a servlet is triggered by an user interaction, i.e. the clicking at an interaction element, and reacts on that interaction by creating a Web page as a response to that user interaction.

As a consequence, the Navigational Transformation is done in the following way. We replace the WAM in the Navigational PIM by a servlet and embed into the doGet-method the slightly modified behavioral model from the Navigational PIM. The doGet-method calls, instead of the WAM, the clicked operation of an InteractionElement of a Node. The behavioral model is modified to invoke a display-operation provided by the servlet, instead that of the WAM.

This transformation from the platform-independent model to the platform-specific model is straightforward and relatively simple. We give two examples for it in the next two sections. The first example covers anchors, links and dynamic page creation; the second example covers the execution of an...
action, which is triggered by pressing a button.

6.2 Servlet-Based PSM for Dynamic Navigation

We use dynamic navigation (see section 3.2) as an example and present the servlet-based Navigational PSM for it in Figure 9.

```java
class ShoppingCartNodeServlet extends OOHDMServlet {
    private Node myNode;
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        String action = req.getParameter("action");
        // Process the action...}
}
```

We introduce a class ShoppingCartNodeServlet which has a reference to the ShoppingCartNode. When a user presses an interaction element of the Web page that is modeled by ShoppingCartNode, the servlet calls the clicked-method of the corresponding InteractionElement of ShoppingCartNode, like that of the DynPageAnchor for dynamic navigation. Now, the behavioral model classes from the PIM execute their methods as defined there, with one exception: whereas a PIM method like traverse calls the display-operation of the WAM, the PSM method is modified to call the display-method provided by the servlet. To make this possible, we modify the clicked-method and the navigate- and traverse-methods by adding and forwarding a reference to the servlet as an additional parameter, and we introduce the class OOHDMServlet from which ShoppingCartNodeServlet is derived.

The class OOHDMServlet, derived from the Java class HttpServlet, contains the method display(theNode: Node) which replaces the display-operation of the WAM. The method gets the associated Page from theNode, and writes it to the response attribute of type HttpServletResponse. The doGet-method has set before this response-attribute to the response parameter named res, so that the content of the response-attribute is displayed as a Web page at the end of the doGet-method call.

```java
class OOHDMServlet extends HttpServlet {
    HttpServletResponse response;
    void display(n: Node) {
        Page p = n.getPage(); //get Web page created from Node
        //write Web page p on response-parameter
        out = response.getWriter();
        while (p.hasMoreElements())
            out.println(p.next());
    }
}
```

The ShoppingCartNodeServlet, derived from OOHDMServlet, has a method doGet. This method analyzes which interaction element the user has pressed; calling the getInteractionElement-method of ShoppingCartNode it gets the corresponding interaction element, like the DynPageAnchor for dynamic navigation, and calls its clicked-method passing a reference to this servlet as a parameter.

```java
class ShoppingCartNodeServlet extends OOHDMServlet {
    private Node myNode;
    public void doGet(HttpServletRequest req,
                       HttpServletResponse res) {
        // Process the request...}
}
```
The transformed behavioral model classes collaborate in the same way as described in section 3. When the doGet-method of ShoppingCartNodeServlet calls the clicked-method of the DynPageAnchor, that method forwards the call to the navigate-method, which calls the traverse-method of the DynPageLink, each time passing a reference to ShoppingCartNodeServlet as a parameter.

The traverse-method calls first the find-method and then the set-method of the target node CDNode so that the target node gets the dynamic page content from the DynEntity associated to it and inserts it into the DynPage that contains already the static HTML page content. When the traverse-method calls eventually the display-method of the servlet with the target node as a parameter, this method writes the page from the DynPage to the response-attribute that references the response-parameter of the servlet. Consequently, the doGet-method of the servlet displays the dynamic Web page that the CDNode has built.

6.3 Servlet-Based PSM for Advanced Navigation

We use advanced navigation (see section 3.2) as another example and present the servlet-based Navigational PSM in Figure 10.

We transform the behavioral model for advanced navigation (see Figure 5) exactly in the same way as the behavioral model for dynamic navigation. The class CDNodeServlet has a reference to the CDNode. When it analyzes the user response and determines that the AddToCartButton has been pressed, it calls the getInteractionElement-method of CDNode to get the corresponding interaction element, like AddToCartButton, and calls its clicked-method passing a reference to "this" servlet as a parameter. As described, the clicked-method calls the action-method 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.

```
addToCart(s:OOHDMServlet) {
  value = this.getField(key);
  shoppingCart.add(value);
}
```

7. Related Work

Many Web application design methods proposed in the last years, like WebML by Ceri, Fraternali, and Paraboschi [C00] and W2000 by Baresi, Garzotto, and Paolini [B00], do not have special constructs to model business processes. Recent proposals like UWE by Koch and Kraus [KKCM03], OO-H by Cachero and Melia [KKCM03], and OOWS by Pastor, Fons and Pelechano [PFP03] have constructs for the modeling of business processes, but do not give a formal definition of the semantics of the model constructs. OOWS captures formal system requirements formally to construct from them the Web application.

8. CONCLUSIONS

We have presented a model-driven architecture for Web applications. It uses the object models of OOHDM enhanced by a behavioral semantics definition as platform-independent model, which is subdivided into two sub-models, a Conceptual PIM and a Navigational PIM. There is a choice of different Conceptual PSMs and Navigational PSMs. The two sub-models are transformed quite independently in a Conceptual PSM and a Navigational PSM. Examples for transformations into two concrete PSMs are given.

Acknowledgements

My thanks for a partial support of the project are due to the International Bureau of the BMBF, Germany, and SeCTIP, Argentina, Bilateral Cooperation with Argentina; and to the Ministerium fuer Wissenschaft und Forschung, Baden-Wuerttemberg.
9. REFERENCES


[GHJV95] E.Gamma, R.Helm, R.Johnson, J.Vlissides: “Design Patterns: Elements of Reusable Object-Oriented Software”. Addison-Wesley, 1995


