A Behavioral Semantics of OOHDM Core Features and of Its Business Process Extension

Hans Albrecht Schmid and Oliver Herfort

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

Abstract. OOHDM models hypermedia-based Web applications by an object model on three layers. Recently, an OOHDM extension by business processes has been proposed. In all cases, the definition includes a formal description of the syntactical aspects and a verbal description of the semantics. In this paper, we give a behavioral definition of the semantics of the OOHDM core features: navigation and advanced navigation; and of the proposed extension by business processes. We derive application-specific model classes from predefined behavioral model classes that have operations with a well-defined semantics. The behavioral model classes collaborate with a Web Application virtual Machine (WAM). The WAM models basic Web-browser characteristics, i.e. HTTP-HTML characteristics. Thus, the semantics of an OOHDM Web application model is precisely defined in an executable way.

1 Introduction

The Object-Oriented Hypermedia Design Method OOHDM by Schwabe and Rossi [SR98] is a modeling and design method for Web applications, which describes hypermedia-based navigation by an object model on three levels, the conceptual level, the navigational level and the interface level. Recently, Schmid and Rossi proposed an extension of OOHDM by business processes [SR02] [SR04]. The definition of OOHDM includes a formal description of syntactical aspects and a verbal description of the semantics.

However, verbal descriptions like that are not always precise, but often vague and open to misunderstandings and doubts. A formal definition of the semantics is required to cope with this problem. Since OOHDM uses object models, it lends itself to a behavioral definition of the semantics in form of the object behavior.

We focus in this paper on a behavioral definition of the semantics of the OOHDM core features, which are navigation and advanced navigation; and of the proposed OOHDM business process extension. OOHDM models a Web application by application-specific classes with a semantics that is given verbally. Our approach is to

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 74–87, 2004.
© Springer-Verlag Berlin Heidelberg 2004

derive these classes from predefined behavioral model classes that have operations with an executable semantics definition. The behavioral model classes, called shortly model classes, collaborate with a Web Application virtual Machine (WAM). The WAM models basic Web-browser characteristics, i.e. HTTP-HTML characteristics. Thus, the semantics of an OOHDM model of a Web application is precisely defined in an executable way.

We present and explain an OOHDM model of a Web shop that includes navigation, advanced navigation and a business process in section 2. Section 3 introduces the Web Application virtual Machine WAM and related classes and services. Section 4, 5 and 6 define the semantics of the OOHDM navigation, advanced navigation and business process constructs by a behavioral model. Section 7 surveys shortly related work.

2 The Web Shop as an Example for an OOHDM Model

We use 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 1); it models abstracted Web pages and the navigation possibilities among them in a navigational schema (see Figure 2); and the presentation aspects of Web pages (which we disregard) in an interface schema.

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, which OOHDM considers just as a classification, indicate the category to which each object class belongs. But for the behavioral model definition, each stereotype indicates the model class an application-specific class is derived from. Note that the classes shown in Figure 1 and 2 do not give a complete application model of the example Web shop, to avoid an overloading with details. The required details will be given in sections 4-6.

The relationship between objects in the conceptual and navigational schema, like that between an entity node CDNode 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.

Pure Navigation

Consider, for example, the navigation possibilities in the CD store. On the left of the navigational schema (see Figure 2), you find entity nodes (which are an abstraction of Web pages) like CDNode 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 CDNode or to the ShoppingCartNode, from the CDNode to the ShoppingCartNode and back, and from one CD to other related CDs by using the “related” link between CDNode’s.

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.

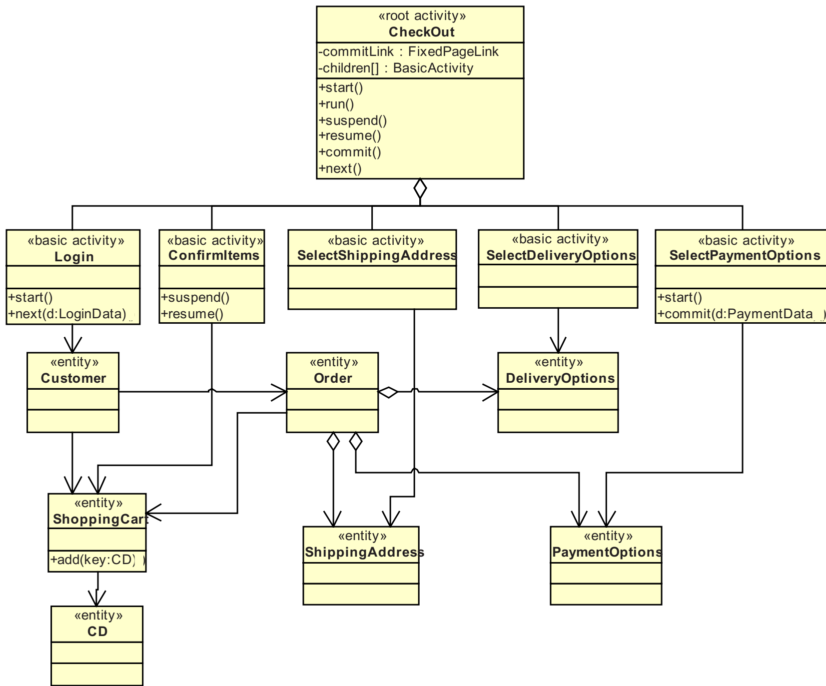

Fig. 1. OOHDM conceptual schema of a Web shop including entities and a business process

Business Processes

An entity object like CD or Customer has a permanent lifetime and state; a process object like CheckOut has a temporary state and no permanent lifetime (see [S99] [SR04]). Processes and activities are modeled in the OOHDM conceptual schema (see Figure 1 top), and activity nodes in the OOHDM navigational schema (see Figure 2 right).

Typically, a business process like CheckOut (see Figure 1 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*, *commit*, *suspend* and *resume*, that allow to execute a business process.

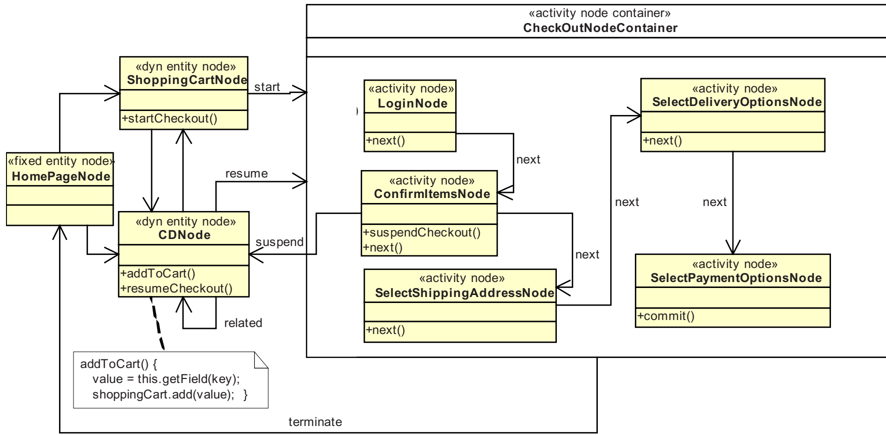

Fig. 2. OOHDM navigational schema of a Web shop including navigation, advanced navigation and a business process

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 2 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 Model Classes and the Web Application Virtual Machine

To define the semantics of application-specific classes like those presented in Figure 1 and 2, we derive them from behavioral model classes with a predefined semantics. We use the UML stereotype "model" to label a model class. The basic model classes are: Entity, RootActivity, and BasicActivity on the conceptual layer; Node and Link on the navigation layer; and InteractionElement on the interface layer. Node, Link and InteractionElement are specialized as described in the next paragraphs and Figure 3. The model classes collaborate with the Web Application virtual Machine, abbreviated WAM. The WAM models basic Web-browser characteristics, i.e. HTTP-HTML characteristics as seen from a Web application.

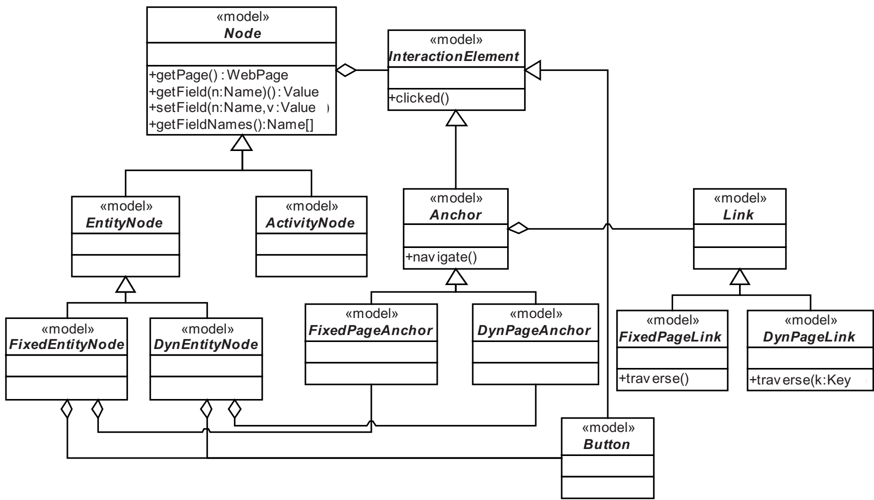

Fig. 3. Specialized behavioral model classes Node, InteractionElement and Link

The model class Node models a Web page abstracting from many of its concrete characteristics, and the model class InteractionElement models interaction elements like anchors and buttons.

The class InteractionElement defines the operation *clicked*() which is called when an end user clicks at the interaction element. We refine the class InteractionElement in Anchor and Button, and Anchor in FixedPageAnchor and DynPageAnchor. Their detailed characteristics are presented in section 4 and 5.

The class Node defines the (abstract) operations *getPage*(): WebPage, *getField*(n:Name): Value, *setField*(n: Name, v: Value), *getFieldNames*(): Name []. It contains an array of InteractionElements. We refine a Node into EntityNode and

ActivityNode: an EntityNode may contain Anchor's and Button's, an ActivityNode only Button's as InteractionElements. The class Link is refined into FixedPageLink with a method *traverse*(), and DynPageLink with a method *traverse*(k: Key).

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.

In the following sections, the behavioral model gives the executable definition of the operations in the form of UML notes in Java. As an alternative to Java, we might use the action semantics of UML 1.5/2.0. Operations defined by the behavioral model are printed in bold fonts to distinguish them from application-specific operations. For lack of space, the behavioral model presented does not include items like error cases.

4 The Behavioral Semantics of Navigation

Navigation allows a user to navigate from a given node to a linked node by clicking at the link. Its main characteristic is that the state of navigation is determined only by the page displayed by the browser and its content [SR04]. The behavioral model of navigation mirrors this characteristic: it shows that navigation does not cause any state changes but that of the current node of the WAM.

We distinguish two kinds of navigation, navigation among Web pages with a fixed content and navigation among Web pages with a dynamic content.

Navigation Among Web Pages with a Fixed Content

Fixed page navigation follows a link to a node without dynamically generated content, like the link from CDNode to HomePageNode 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, FixedPageAnchor, and FixedPageLink; which are instantiated and configured to work together. These model classes have executable definitions of the methods *clicked*, *navigate* and *traverse*.

A node like the CDNode contains a FixedPageAnchor instance, which references a FixedPageLink instance, which, in turn, references the target node of the link. Each reference is set by a constructor 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 FixedPageAnchor, which calls the *traverse*-operation of the FixedPageLink. 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.

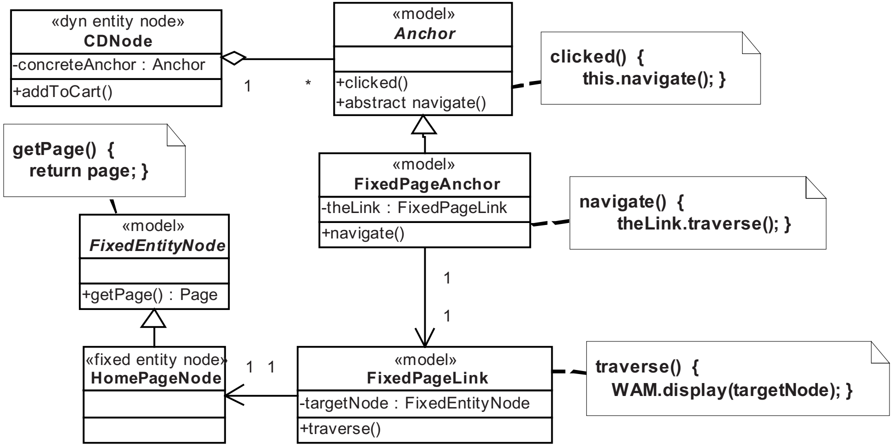

Fig. 4. Static navigation from CDNode to HomePageNode

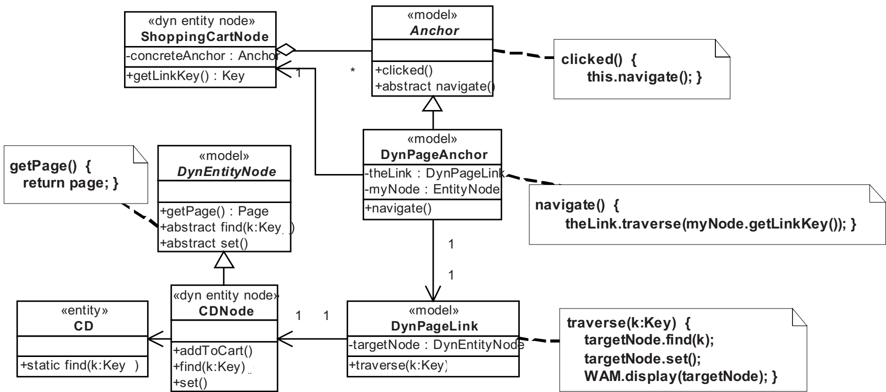

Fig. 5. Dynamic navigation from ShoppingCart Node to CDNode

Navigation to Web Pages with a Dynamically Generated Content

Navigation to a dynamic page is similar to fixed page navigation, except for the key required to identify the dynamic content (see Figure 5). The differences are:

- A source node of a dynamic link has an operation *getLinkKey* that returns a key identifying the dynamic content of the target node.

- 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) dynamic content into the attribute fields.

- The class DynPageAnchor has a *navigate*-operation that fetches the key of the dynamic content from the source node, called myNode, and passes it as a parameter with the call of the *traverse*-operation. The *traverse*-method 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, it calls the *display*-operation of the WAM with the target node as a parameter.

5 The Behavioral Semantics of 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. Consequently, the Web application state is composed by the state of the current node displayed by the browser, and by the state of the application objects. The behavioral model of advanced navigation shows that not only the current node of the WAM, but also application domain objects may change their state.

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.

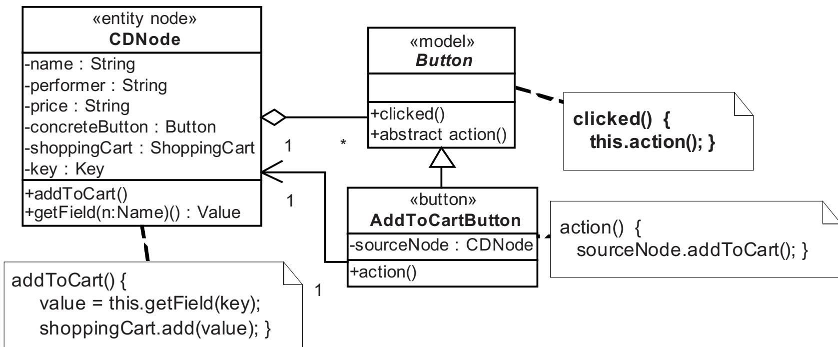

Fig. 6. Triggering the execution of the atomic *addToCart*-action by a button

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.

6 The Behavioral Semantics of Business Processes

The main characteristics of a business process (see [SR04]) are that it:

- drives the user through its activities. This means it defines the set of activities to be executed, and the possible control flow among them.
- keeps its state internally. The state can be changed only by the process itself on a user interaction, but not by the user pressing a browser button.

The behavioral semantics of a business process mirrors these characteristics. The behavioral model defines the semantics of process state transitions, which are represented by process-related edges like *start* and *next* in the navigational schema (see Figure 2): it shows that a business process changes its state only when a process state transition is triggered by the user, and that, in general, the process, and not the user, selects the next activity for execution.

To make the behavioral model not unnecessarily difficult to understand, we make the restriction (not made by [SR04]) that there are only two levels of process execution, i.e. a process consists of a root activity with basic activities as children. The behavioral model defines the classes RootActivity and BasicActivity, with operations like *start*, *run*, *next*, *commit*, *suspend*, and *resume* with a predefined semantics, given by final methods or inherited from class Thread. Application-specific activity classes are derived from them.

A business process is executed independently from navigation in an own thread, so that it can preserve its state when suspended, and be resumed at the point of the suspension. We use Java threads to model the business process threads and navigation thread. The model class RootActivity, derived from Thread, inherits Thread-operations to *start*, *suspend*, *resume* and complete a business process thread.

6.1 Starting and Terminating a Business Process and an Activity

A business process is started from an action method of an entity node. E.g., the *startCheckout*-method of the ShoppingCartNode (see Figure 2 and 7), triggered by a user pressing a StartButton, starts the Checkout process and suspends the navigation thread, calling *suspend* of the Thread under which it runs. The mechanics of buttons and invocation of the action-method is exactly the same as described in section 5; so we do not present it again here and in the following examples and figures.

The behavioral model for starting and terminating a business process is shown in Figure 7. The *start*-method that Checkout inherits from Thread calls the predefined *run*-operation of the business process thread. The final *run*-method of the model class RootActivity, which realizes that operation, is a template method [GHJV95] that is inherited by the application-specific Checkout root activity. It calls the application-specific methods *startHook* for initialization and *getNextActivity* to select the basic activity to be executed, sets this activity, like Login, as current (-ly executed) activity, and starts it.

Starting a Basic Activity

The predefined *start*-operation of the model class BasicActivity initializes the content data of the associated activity node and displays it as Web page. It is a template method that is inherited by an application-specific activity like Login. The *start*-method calls the application-specific *startHook*-method. This method usually gets the data to be presented to the user from the application domain objects and puts them into the attribute fields of the associated ActivityNode; but the Login activity presents no data in the LoginNode. Then the *start*-method calls the *display*-method of the WAM passing the activity node like LoginNode as a parameter, so that the WAM displays it as a Web page.

Completing a Basic Activity

All activity nodes of the Checkout process have a next-button to complete the activity and go to the next one, except for the SelectPaymentOptionsNode which has a commit-button, since it completes the SelectPaymentOptions activity and then the Checkout process.

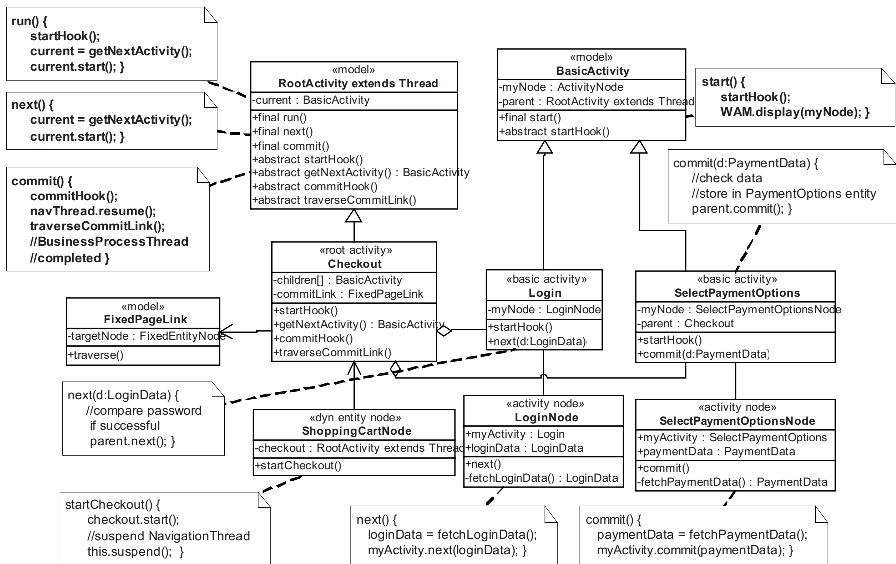

Fig. 7. Starting the CheckOut process from the ShoppingCartNode and completing it from the SelectPaymentOptionsNode.

For example, the LoginNode has a next button which triggers its *next*-method. This method fetches the data from the LoginNode which the user has entered during the Login activity, and passes them as parameters with the call of the method *next*(d:LoginData) of the Login activity. This method compares the entered user name and password with the associated Customer object, and calls, if the check is successful, the *next*-method of Checkout. This method is a template method inherited from the model class RootActivity; it selects the activity to be executed next by calling the

application-specific hook method *getNextActivity*, sets this activity as current (-ly executed) activity and starts it.

Terminating the Checkout Business Process

A business process like Checkout is terminated when it has completed its work, that is when the last of its child activities, SelectPaymentOptions, has completed its work and Checkout has completed the customer order and sent it off.

The SelectPaymentOptions activity completes its work when the commit-button triggers the *commit*-method of the SelectPaymentOptionsNode (see Figure 7 bottom right). This method fetches the payment options data, which the user has entered during the SelectPaymentOptions activity, and passes them as parameters with the call of the *commit*(d: PaymentData) method of the SelectPaymentOptions activity. This method checks the entered data and stores them in the PaymentOptions object. Then it calls the *commit*-method of its parent activity Checkout.

The Checkout activity inherits the template method *commit* from the model class RootActivity. The *commit*-method calls the application-specific *commitHook* to complete the processing of the order before it resumes the navigation thread (with the Thread *resume*-operation) and calls the application-specific traverseCommit Link-method. This method calls the *traverse*-operation of the FixedPageLink to the HomePageNode, so that the WAM presents the home page to the user. The business process thread is completed with the end of the *commit*-method.

6.2 Suspending and Resuming a Business Process

A business process like Checkout may be suspended only from activity nodes where this is provided for by a Web application designer. E.g., the ConfirmItemsNode has a method *suspendCheckout* that is triggered by a user pressing a NavigateToCD-button. The behavioral model for suspending and resuming a business process is given in Figure 8.

The *suspendCheckout*-method of ConfirmItemsNode calls the *resume*-operation of the navigation thread, and the *traverse*-method of the suspendLink in order to navigate to the CDNode, before it calls *suspend* of ConfirmItems to suspend the business process.

The ConfirmItems activity inherits the *suspend*-method, which is a template method, from the model class BasicActivity. This method calls the *suspendHook*, which saves the activity state if required, as e.g. after user inputs during the activity, and stores it temporarily in the activity state, and then the *suspendBP*-method of the Checkout activity. The *suspendBP*-method, which the Checkout activity inherits from the model class RootActivity, calls the application-specific *suspendHook* that performs application-specific actions before the suspension, and then the Thread *suspend*-operation to suspend the business process thread.

Resuming the Checkout Business Process

A business process like Checkout may be resumed only from entity nodes where this is provided for by a Web application designer. E.g., CDNode has a *resumeCheckout*-

method that is triggered by a user pressing a ResumeCheckout-Button during the temporary navigation. Figure 8 gives the behavioral model for resuming a business process, which suspends the navigation thread.

The *resumeCheckout*-method of CDNode calls *resume* of the Checkout process (if Checkout is not suspended, an exception is returned) and then the *suspend*-method that the navigation thread inherits from Thread. The *resume*-method, which Checkout has inherited via the model class RootActivity from Thread, resumes the business process thread which has been suspended in the *suspendBP*-method. When this method is resumed, it calls the application-specific *resumeHook*, which may e.g. restore some state if required. Finally, it calls the *resume*-method of the basic activity like ConfirmItems that was active when the business process was suspended.

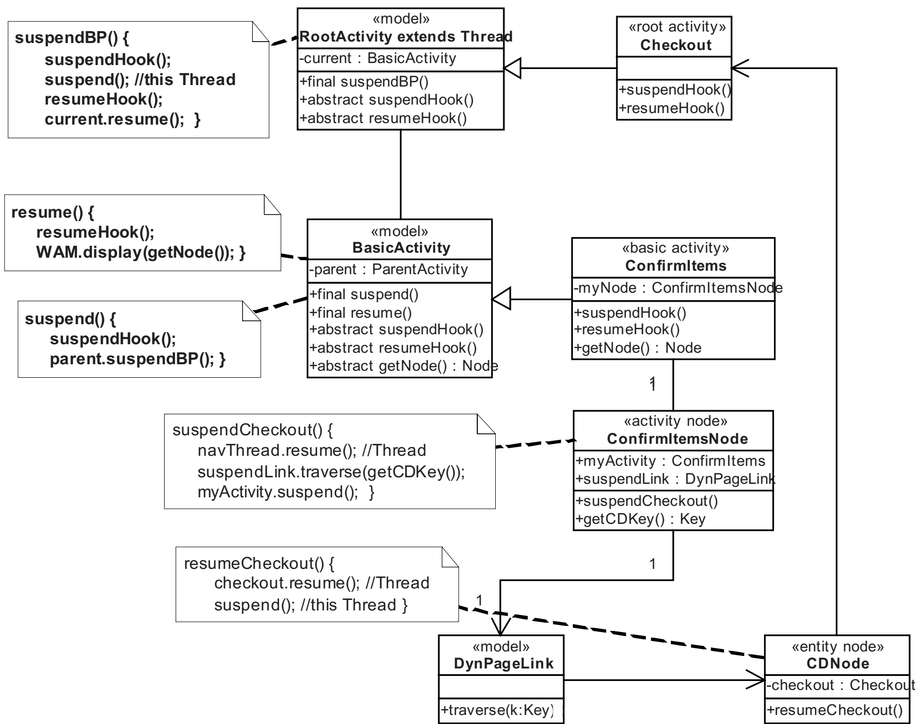

Fig. 8. Suspending the CheckOut process from ConfirmItemsNode and resuming it from CDNode

ConfirmItems has inherited the *resume*-method from the BasicActivity class. The *resume*-method calls first the application-specific *resumeHook*. This method restores the data, which were presented to the user and saved when the activity was suspended, from the application state or temporary objects, and puts them into the attribute fields of the associated ActivityNode, like ConfirmItemsNode. The *resume*-method calls then the *display*-method of the WAM passing the ConfirmItemsNode as

a parameter so that the WAM displays the ConfirmItemsNode as a Web page, and the user can go on with the execution of the Checkout process.

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 functional system requirements formally to construct from them the Web application.

8 Conclusions

We have given a behavioral definition of the semantics of OOHDM and of its business process extension. The definition is not complex and easy to understand. One reason for that seems to be that the OOHDM designers did an excellent work: the OOHDM conceptual and navigational layers, and a few aspects of the interface layer, seem to focus exactly on all important aspects of a Web applications, without giving unnecessary details nor a description of platform- or technology dependent aspects. Another reason seems to be that OOHDM as an object-oriented approach lends itself to a behavioral definition of the semantics.

We have defined a Web Application virtual Machine WAM. Application-specific classes and behavioral model classes collaborate with it. A version of the WAM that neglects the Web page layout given by the OOHDM interface schema, is easy to implement. With that WAM version, both the behavioral definition of the OOHDM semantics and an OOHDM model of a Web application can be directly executed and tested.

Another very promising aspect is the use of the behavioral OOHDM semantics definition for a model driven architecture approach. For example, the entities from the conceptual layer may either model e.g. Corba or Enterprise JavaBean application objects from an existing back-end application, or they may conceptualize non-existing lightweight entities that provide just an interface to a database management system. In the first case, a generator just generates code to invoke the existing application objects from the servlets that realize the OOHDM nodes, whereas in the second case, a generator generates servlets that realize the OOHDM nodes and embody the database access provided by the lightweight entities.

Acknowledgements. Our thanks for a partial support of the project are due to the International Bureau of the BMBF, Germany, Bilateral Cooperation with Argentina, and SeCTIP, Argentina; and to the Ministerium fuer Wissenschaft, Forschung und Kunst, Baden-Württemberg.

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

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

[KKCM03] N.Koch, A.Kraus, C.Cachero, S.Melia: “Modeling Web Business Processes with OO-H and UWE“. IWWOST 03, Proceedings 3rd International Workshop on Web-Oriented Software Technology, Oviedo, Spain, 2003

[PFP03] O. Pastor, J. Fons, V. Pelechano: “OOWS: A Method to Develop Web Applications from Web-Oriented conceptual Models“. IWWOST 03, Proceedings 3rd International Workshop on Web-Oriented Software Technology, Oviedo, Spain, 2003

[S99] H.A. Schmid: ”Business Entity Components and Business Process Components”; Journal of Object Oriented Programming, Vol.12, No.6, Oct. 99

[SR02] H.A. Schmid, G. Rossi: “Designing Business Processes in E-Commerce Applications“. In E-Commerce and Web Technologies, Springer LNCS 2455, 2002

[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