

# Modeling Web Business Processes with OO-H and UWE<sup>1</sup>

NORA KOCH AND ANDREAS KRAUS  
Ludwig-Maximilians-Universität München.  
Germany

CRISTINA CACHERO AND SANTIAGO MELIÀ  
Universidad de Alicante, Spain

---

Business processes, regarded as heavy-weighted flows of control consisting of activities and transitions, play an increasingly important role in Web applications. In order to address these business processes, Web methodologies are evolving to support its definition and integration with the Web specific aspects of content, navigation and presentation.

This paper presents the model support provided for this kind of processes by both OO-H and UWE. For this support both approaches use UML use cases and activity diagrams and provide appropriate modeling extensions. Additionally, the connection mechanisms between the navigation and the process specific modeling elements are discussed. As a representative example to illustrate our approach we present the requirements, analysis and design models for the amazon.com Website with focus on the checkout process. Our approach includes requirements and analysis models shared by OO-H and UWE and provides the basis on which each method applies its particular design notation.

**Categories and Subject Descriptors:** H.5.4 [*Information Interfaces and Presentation*]: Hypertext/Hypermedia – Architecture; Navigation; User Issues; D.2.2 [*Software Engineering*]: Design Tools and Techniques – *Object-oriented Design Methods*; D.2.1 [*Software Engineering*]: Requirements Specification – *Methodologies*; I.6.5 [*Computing Methodologies*]: Simulation and Modeling – *Model Development*

**General Terms:** Design, Experimentation, Human Factors

**Additional Key Words and Phrases:** Web Engineering, UML, Visual Modeling, Process Modeling, UML Profile

---

## 1. INTRODUCTION

Business processes, regarded as heavy-weighted flows of control consisting of activities and transitions [UML 2003], have always paid an important role in software development methods, to the point that many process proposals include the definition of an explicit view (the *process view*) in order to address their complexity. However, such processes have been only tangentially tackled in most existing Web Application modeling approaches [Retschitzegger & Schwinger 2000]. This reality is partly due to the fact that most of these methods were born with the aim of successfully modeling Information Systems (IS) [Baresi et al. 2001, Schwabe et al. 2001, Ceri et al. 2002, De Troyer & Casteleyn 2001, Gomez et al. 2001, Koch & Kraus, 2002], which *'store, retrieve,*

---

<sup>1</sup> This work has been partially supported by the European Union within the IST project AGILE – Architectures for Mobility (IST-2001-32747), the German BMBF-project GLOWA-Danube, and the Spain Ministry of Science and Technology, project number TIC2001-3530-C02-02.

Authors' addresses: Nora Koch and Andreas Kraus, Ludwig-Maximilians-Universität München, Germany, <http://www.pst.informatik.uni-muenchen.de>, {kochn,krausa}@informatik.uni-muenchen.de. Cristina Cachero and Santiago Melià, Universidad de Alicante, España, <http://www.ua.es>, {ccachero,santi}@dlsi.ua.es.

*transform and present information to users. [They] Handle large amounts of data with complex relationships, which are stored in relational or object databases'* [Korthaus 1998]. Therefore, Web modeling approaches have intensively worked on (1) the definition of suitable constructs to address the user navigation through the domain information space and (2) the inclusion of mechanisms to model the change in such information space through the invocation of synchronous retrieval and update operations. In contrast, requirements posed on modern Web applications imply to regard them not only as Information Systems but also as Business Systems, that is, applications centered on goals, resources, rules and, mainly, the actual work in the business (business processes). Workflow management systems, which have proven successful for the definition and control of these processes, fall short however when faced to the problem of defining rich business-enabled Web interfaces that, aware of the underlying workflow, support and guide the user through it, preserving at the same time the hypertext flexibility.

The hypermedia community, conscious of this gap, has been working for some time on the extension of Web modeling methods (which are specially suited to deal with the complexity of Web interfaces) with new mechanisms that permit the definition and integration of lightweight business processes with the rest of the views. This extension may be done following at least two different approaches: on one hand, traditional content, navigation and/or presentation models may be enriched to capture this workflow. Examples in this sense include Araneus2 [Atzeni & Parente 2001], which defines the mechanisms to allow the grouping of activities into phases, Wisdom [Nunes & Cunha 2000], which proposes a UML extension that includes the use of a set of stereotyped classes, or WebML [Brambilla et al. 2002] which enriches the data and the hypertext models to define lightweight web-enabled workflows. On the other hand, additional models may be defined, and its connection with the pre-existing content, hypertext and/or presentation views established. This has been the approach jointly followed by UWE and OO-H, as we will present in this paper.

Our aim in this paper has been therefore to find a common set of modeling concepts that suffices to define sound, non-trivial business processes and that could be equally useful in other existing methodologies. In order to define the necessary constructs and modeling activities, we have decided to adhere to well known object-oriented standards, namely to the semantics and notation provided by UML. Using UML to model business processes is not new; authors like [Nunes & Cunha 2000, Markopoulos 2000] have already acknowledged its feasibility and excellence. From the set of modeling techniques provided by the UML, the activity diagram is the most suitable mechanism to model the business workflow, and so has been adopted by both OO-H and UWE to define the different processes. In this diagram, activity states represent the process steps, and transitions capture the process flow, including forks and joins to express sets of activities that can be processed in arbitrary order.

In order to reach our goal, this work is organized as follows: Sections 2 and 3 present the analysis steps, equal for both methodologies. Section 4 and Section 5 describe the design steps for OO-H and UWE, respectively. Last, Section 6 outlines conclusions while Section 7 proposes future lines of research. In order to better illustrate the whole approach, a simplified view of the well-known Amazon checkout process (<http://www.amazon.com>) is going to be employed all along the paper.

## 2. THE ROLE OF BUSINESS PROCESSES IN REQUIREMENT ANALYSIS

The inclusion of a business process in any Web modeling approach affects every stage of development, from requirements analysis to implementation. Regarding requirements analysis, whose goal is the elicitation, specification and validation of the user and customer needs, this activity includes the detection of both functional and non-functional requirements, both of which may get affected by process concerns.

Although there is a lack of a standardized process supporting requirements analysis, best practices in the development of general software applications provide a set of techniques. A recent comparative study [Escalona & Koch 2003] about requirements engineering techniques for the development of Web applications showed that use case modeling is the most popular technique proposed for the specification of requirements while interviewing is the most used technique for the capture of those requirements. These results are not surprising; traditional software development requires interviewing as an intuitive and widely used procedure to guide a “conversation with a purpose” [Kahn & Cannell 1957], and use cases are a well known approach for graphical representation and description of requirements suggested by [Jacobson et al. 1992]. Use case modeling is a powerful formalism to express functional requirements of business intensive – Web or non-Web – applications.

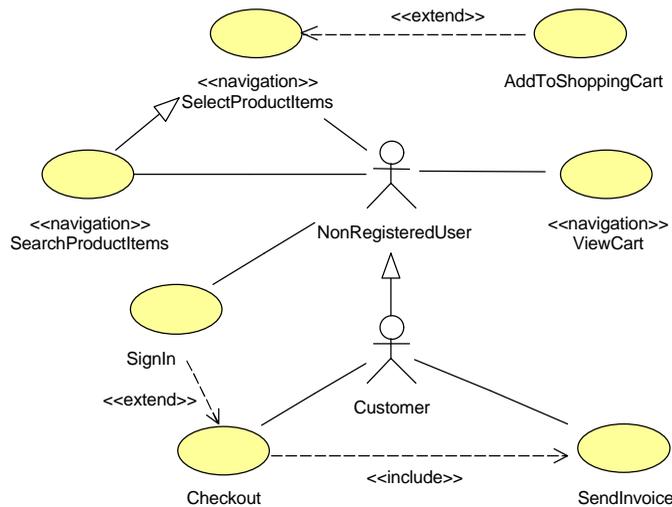
In this sense, OO-H and UWE are object-oriented approaches (partially and completely based on UML, respectively) and both of them include the use case modeling technique to gather the requirements of Web applications. To define a use case model the first step is to identify actors and the corresponding use cases of the application. Such a use case model usually includes a use case diagram which is usually enough to describe the functionality of simple systems, such as of Web information systems. On the other hand, Web applications including business processes require a more detailed description of these – more complex – sequences of actions. In order to address this additional complexity as it is shown in the next section, both approaches propose the use of UML activity diagrams.

In our running example we have identified two actors that play a relevant role in the checkout process: the *NonRegisteredUser* and the *Customer*. The non-registered user can – among other activities – search and select products, add products to the shopping cart and login into the *Amazon Web application*. The *Customer* inherits from the *NonRegisteredUser* and is allowed among other things (after logged-in) to start the checkout process.

Fig. 1 presents a partial view of the use case diagram corresponding to the *Amazon Web application*. For the sake of simplicity, in this diagram we have centered on the use cases that are directly related to the selection of items and the checkout process, therefore ignoring others such as, just to name a few, *AddToWishList*, *CheckOrder* or *ReturnItems*, which are however also relevant tasks from the user point of view.

In this diagram we can observe how a *NonRegisteredUser* may select product items. Such selection may be performed using a system search capability, which is modeled by means of an inheritance relationship between the use cases *SelectProductItems* and *SearchProductItems*. Also, this user may decide to add any selected product to his

shopping cart. This fact is modeled by means of an «extend» dependency between the use case *AddToShoppingCart* and the *SelectProductItems* use case.



**Fig. 1. Use Case Diagram of the Checkout Process in [www.amazon.com](http://www.amazon.com)**

At any time, the user may decide to *ViewCart*, in order to check the items included so far in his shopping basket. Also, he could decide to personalize her view, for what he would have to *SignIn*. Furthermore, only a signed-in user may proceed to checkout. This situation is again modeled by means of an «extend» dependency between the use cases *SignIn* and *Checkout*. The completion of the checkout process implies the sending of a notification with the invoice associated with the purchase. We have modeled this action as a use case that is related («include» dependency) to the *Checkout* use case. The *Customer* may also wish to be sent an additional invoice at any time after the purchase; in Fig. 1 this fact is captured by means of an association between the actor *Customer* and the *SendInvoice* use case.

If we now analyze the inner flow of control of each defined use case in the context of the *Amazon Web application*, we may note how some flows are trivial from the business point of view, as they only express navigation activities. For this kind of use cases, we propose the use of a «navigation» stereotype, as defined in [Baresi et al. 2001]. Others, on the contrary, imply a complex flow of control, and require further refinements, as we will show next.

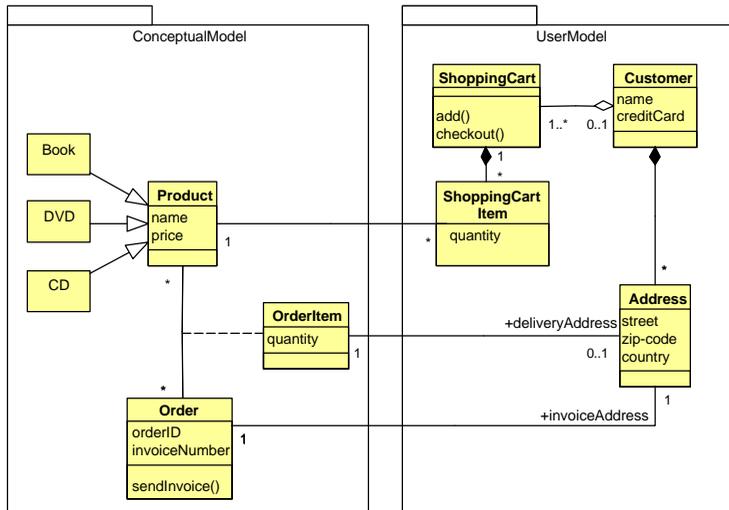
### 3. ANALYSIS PHASE IN PROCESS-AWARE WEB APPLICATIONS

Once the requirements have been clearly stated, both in OO-H and UWE the next step consists on the analysis of the problem domain. This analysis phase has traditionally involved in both methods the definition of a conceptual model reflecting the domain structure of the problem. This model however does not provide the mechanisms to specify process concerns. That is the reason why we have included a new model, the

*process model* that enriches this analysis phase. Next we will show how these models can be applied to our running example.

### 3.1. CONCEPTUAL MODEL

The definition of a conceptual model by means of a UML class diagram is a common feature in most Web modeling approaches, including UWE and OO-H. Back to our example, we have defined a common conceptual model, materialized in a UML class diagram that is depicted in Fig. 2.



**Fig. 2. Conceptual Model of the Checkout Process in [www.amazon.com](http://www.amazon.com)**

This class diagram is made up of two packages: the User Model and the Conceptual Model. The first one includes the structures directly related to the individual user, such as the *ShoppingCart* or the different *Addresses* provided by each *Customer*. The Conceptual Model on the other hand maintains information related to domain objects such as *Products* and *Orders*. Note how this diagram is a simplification of the actual Amazon domain model, and reflects only a possible subset of constructs that are needed to support the *checkout* process. In this diagram we can observe how a customer (which may be anonymous before logging-in in the system) has a *ShoppingCart*, which is made-up of *ShoppingCartItem*s (each one associated with a *Product*, which may be a *Book*, a *DVD* or a *CD*, just to name a few). On the other hand each customer has a set of predefined *Addresses* that, once the order has been created, are used both to send the different items and to set the invoice address. When the customer decides to *checkout*, the system creates a new *Order* and converts the *ShoppingCartItem*s into *OrderItem*s. When the order is finally placed, an *invoiceNumber* is associated to the order.

The conceptual model is not well suited to provide information on the underlying business processes that drive the user actions through the application. For this reason, OO-H and UWE have included a *process model* that is outlined in the next two sections.

### 3.2. PROCESS MODEL

Process modeling (also called task modeling) stems from the Human Computer Interaction (HCI) field. Different UML notations have already been proposed for process modeling. Wisdom [Nunes & Cunha 2000] is a UML extension that proposes the use of a set of stereotyped classes that make the notation not very intuitive. Markopoulos [Markopoulos 2000] instead makes two different proposals: an UML extension of use cases and another one based on statecharts and activity diagrams. Following this last trend, we have opted to use activity diagrams, due to their frequency of use and their flexibility to model flows.

An activity diagram is a special case of a state diagram in which all (or at least most) of the states are actions or subactivity states and in which all (or at least most) of the transitions are triggered by completion of the actions or completion of the subactivities in the source states. The entire activity diagram is attached (through the model) to a UML classifier, such as a use case, or to a package, or to the implementation of an operation [UML 2003]. The UML modeling elements for a process model are activities, transitions and branches. Activities represent atomic actions of the process and they are connected with each other by transitions (represented by solid arrows) and branches (represented by diamond icons). The branch conditions govern the flow of control and in the analysis process model they can be expressed in natural language.

As stated before, both OO-H and UWE use activity diagrams to complement the domain model and define the inner flow of control of non trivial use cases. In Fig. 3 an activity diagram representing the simplified flow of control of the Amazon Checkout process (depicted as a non-navigational use case in Fig. 1) is presented.

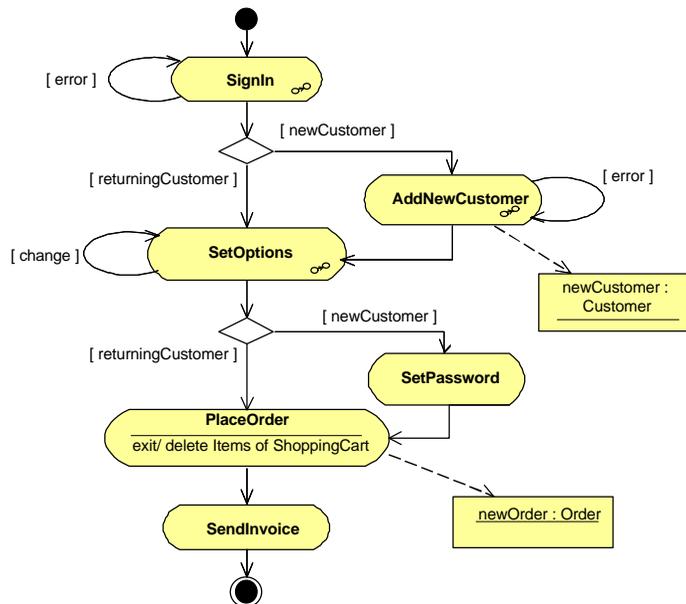
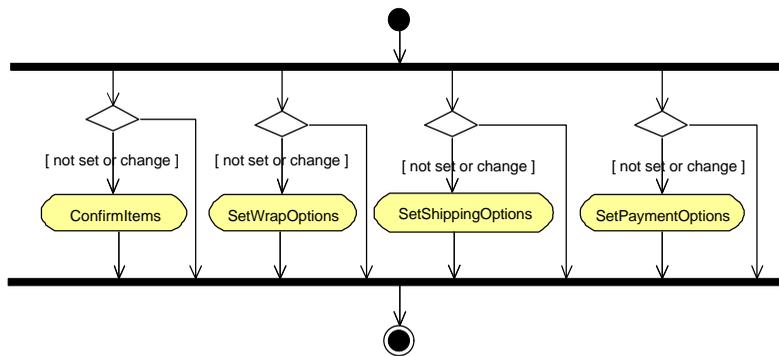


Fig. 3. Process Model of the Checkout Process in [www.amazon.com](http://www.amazon.com)

In this diagram we can observe how a *SignIn* subactivity starts the process. Next, depending on whether the user is new in the system or not, he can be added to the system or directly driven to the *SetOptions* subactivity state that permits the user to establish every purchase option (see ). Once this activities has been completed, the user may set his password (only if he is a new customer), place the order and be sent an invoice with the purchase details.

Subactivity states, as shown in the diagram of Fig. 3, express the hierarchical decomposition of a process. A subactivity state invokes another activity diagram. When a subactivity state is entered, the nested activity graph is executed. A single activity graph may be invoked by many subactivity states meaning that activity diagrams can be (re-)used within the context of different processes and sub processes (i.e. subactivities). In our example, Fig. 4 shows the flow of control of the *SetOptions* subactivity state represented by a UML activity diagram. This diagram includes activities for the user to enter shipping and payment options, wrapping options and confirm the cart items before placing the order. In the checkout process only options not already set before e.g. in previous checkouts or those that the user explicitly wants to change are triggered in the process context.



**Fig. 4. Activity Diagram of the SetOptions Process in [www.amazon.com](http://www.amazon.com)**

In Fig. 3 we can also observe how the *use* and *extend* dependencies defined in the use case diagram of Fig. 1 influence the flow of control of the process, and are materialized in the inclusion of complementary activities (see e.g. *SendInvoice*) and subactivity states (e.g. *SignIn*) that, as suggested by such dependencies, play a role in the definition of the checkout process.

Finally, the activity diagram associated with a given Web-aware business process can also be enriched with object flows indicating objects that are relevant at analysis level as *input* and *output* to crucial activities. In the example a *new Customer* is created as result of the *AddNewCustomer* activity and a new *Order* object is created as result of the *PlaceOrder* activity.

Once this analysis model has been defined, at least two approaches can be followed:

- The definition of a navigation model that is driven by a (refined) process flow model. This tight integration between process and navigation expresses the

interplay between the user interface design and the steps inferred from the process definition.

- The definition of a navigation model that is enriched to reflect a set of integration points, that is, points in which the user may leave the navigation view to enter a process design view.

Next, we will show how OO-H and UWE, each implements one of these approaches, and how in spite of this fact, analysis models are still fully reusable and a common ground for discussion.

## 4. DESIGN OF WEB BUSINESS PROCESS WITH OO-H

OO-H [Cachero 2003; Gomez et al. 2001] is a generic approach, partially based on the Object Oriented paradigm, that provides the designer with the semantics and notation necessary for the development of personalized Web-based interfaces. Like many other approaches, the OO-H modeling process is driven by the set of identified user requirements, and explicitly supports the definition of different user interfaces depending on the actor that is accessing the application. The whole method is supported by the Case tool VisualWADE, a modeling environment that includes a set of model compilers to provide automatic code generation capabilities.

From the six complementary views included in OO-H (requirement, conceptual, process, navigation, presentation and architectural), in this article we will center on the process view. This process view is based, as stated above, on a set of activity diagrams that supplement the information contained in the domain model. In order to make this connection between both models explicit, both class and activity diagrams are refined during the design phase, and the correspondence between the process constructs and the domain constructs is set. On this refined process view, a set of mapping rules can be applied in order to get a default navigation view and assure the traceability between both models, as will be explained in section 4.2.

### 4.1. PROCESS MODEL REFINEMENT

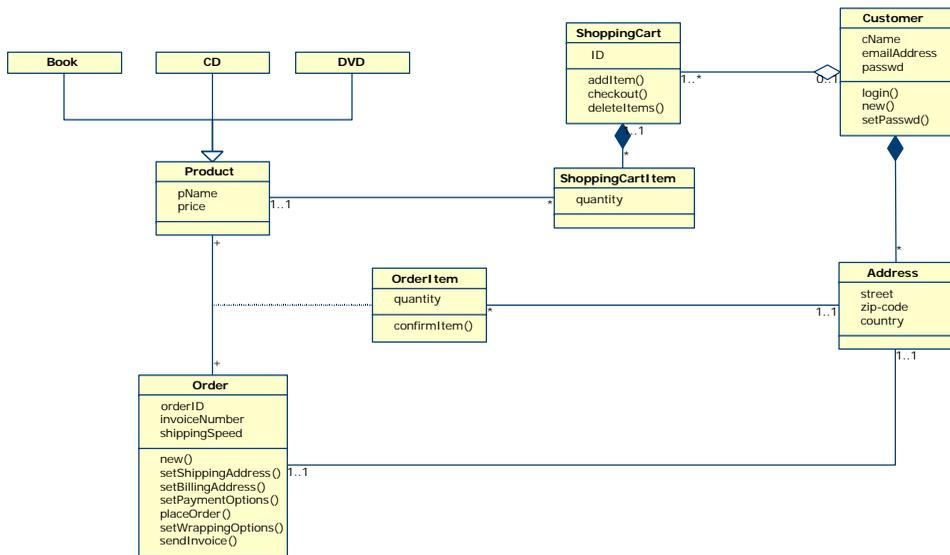
OO-H bases the process refinement on the concept of *service*, regarded as an interface whose purpose is *collect a set of operations that constitute a coherent service offered by classifiers*, and so *provide a way to partition and characterize groups of operations* [UML 2003].

Services in OO-H can be classified according to several orthogonal characteristics, among which we outstand (1) synchronicity, (2) activating agent, (3) granularity (number of operations supporting the service) and (4) transactionality [Cachero 2003]. OO-H centers on synchronous, user-activated services. Inside this group, OO-H is well suited to provide an arbitrary complex interface to single (either transactional or not) services, where by *single* we mean services supported by exactly one underlying class operation. That is the case of Create, Delete and Update operations, which are at the core of typical single, transactional (ACID) services.

On the other hand, the definition of an interface for a compound service, which involves more than one domain operation, presents special challenges from the user interface

modeling point of view. In this kind of services, the user interface is responsible for guiding the user through the different domain operations following a predefined flow of control, which may involve both activity and information dependencies. A *checkout* service such as the one included in Amazon is, from the OO-H point of view, a compound, non-transactional service. For this service, a possible activity diagram representing the system flow of control has been already presented in Fig. 3.

In order to further refine the analysis process view, we must take into account detailed class properties. A new class diagram with a more exhaustive list of attributes and methods is shown in Fig. 5. Note how this diagram can be regarded as a possible evolution of the one presented in Fig. 2.

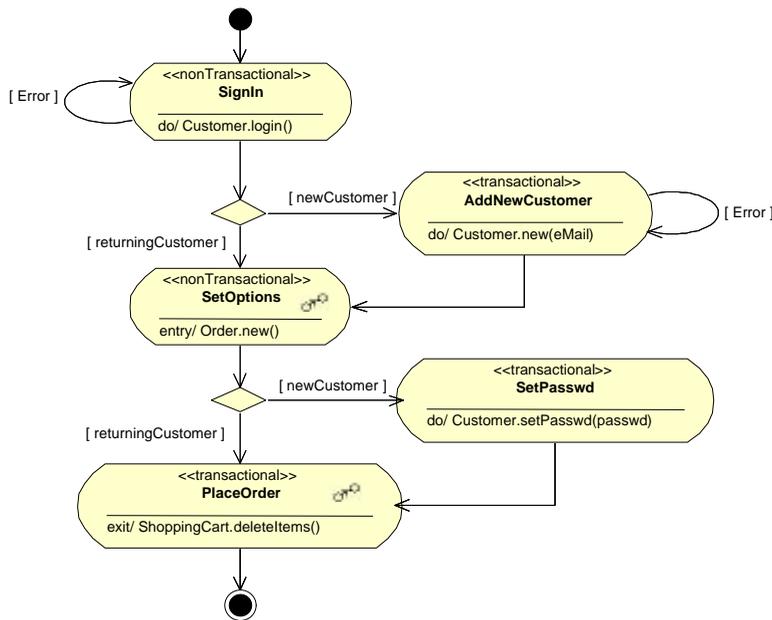


**Fig. 5. OO-H Refined Class Diagram of the Checkout Process in [www.amazon.com](http://www.amazon.com)**

Taking into account the underlying operations, it is possible to construct a more detailed activity diagram, such as the one presented in Fig. 6. In this figure we observe several examples of refinements allowed in OO-H at this level of abstraction:

- Some subactivity states may be redefined as call states, implying that a single operation (at most) gives them support. That is the case of the *SignIn* subactivity state (see Fig. 3) that has been now redefined as a call state.
- Some call states may be merged under a common subactivity state. This feature is specially useful when a transaction is detected which involves several call states. In our example, we have considered that *PlaceOrder* and *SendInvoice* are related activities (in the checkout process the invoice is automatically sent after the order has been placed), and that an error while sending the invoice implies that the order cannot be placed, because the user would lack the necessary notification. Therefore we have defined a transactional subactivity state that includes both call states (see Fig. 6).

- Call states may be complemented (if necessary) with information regarding the domain operation that gives them support in a *do* section. For example, the transactional activity *SetPasswd()* states in the refined activity diagram that this activity is supported by the operation *setPasswd(in passwd:String)*, which belongs to the *Customer* class. Note how this fact could also have been modelled by means of swimlanes added to the diagram.
- A new «transactional» stereotype must be applied to the different activities. This stereotype reflects the transactional character of both call states and subactivity states. A transactional activity/subactivity state presents the classical ACID properties. Furthermore, transactional subactivity states imply that every underlying activity or subactivity state belongs to the same transaction. On the contrary, a non-transactional subactivity state does not pose any requirement over the elements included. Back to our example, the *SignIn* activity has been defined as non-transactional. In our approach this fact implies that the activity does not need logic support, as it might be modelled with the aid of filters and navigation constructs, as we will show in section 4.2



**Fig. 6. OO-H Refined Activity Diagram of the Checkout Process in [www.amazon.com](http://www.amazon.com)**

For the sake of simplicity, in Fig. 6 we have hidden other possible refinements, such as for example the set of OCL guard conditions that may be associated with the transitions, the OCL formulae that may be associated to non-transactional activities or the detailed flow of objects among activities and/or subactivity states, which would also be relevant at this stage of the model and from which it would be possible to infer certain parameter dependencies during the invocation of the underlying methods if necessary.

## 4.2. DEFAULT NAVIGATION MODEL

As we have stated before, OO-H redefines call states so that they can be mapped to underlying domain constructs. In order to support this mapping, the UML metamodel must be conservatively extended to capture this connection between OO-H activities (a specialization of the UML activity construct) and conceptual operations and/or classes. The reason for this connection is that, as we will see next, in this way it is possible to automatically generate a default navigation view that not only speeds up the development cycle but also assures the interface guidance and support to this process.

The navigation model in OO-H is defined by means of a Navigation Access Diagram (NAD). This diagram is made up of collections (depicted as an inverted triangle and which capture in OO-H the menu concept), navigation targets (navigation subsystems depicted with a package symbol), navigation classes (views on conceptual classes, depicted with the class symbol) and navigation links (which describe the paths the user may follow through the system and that are depicted with the arrow symbol). Navigational links, being a central construct in OO-H and the core of the navigation view, have several relevant characteristics associated:

- Type. It can be set to (1) *requirement link*, which determines the beginning of the user navigation through the diagram, (2) *traversal link*, which defines navigation paths between information structures or (3) *service link*, which provides an arbitrarily complex user interface to assign values to the underlying *in* operation parameters and/or define the visualization of the operation results (*out* parameters).
- Activating Agent: can be set to *user* (depicted as a solid arrow) or *system* (depicted as a dotted arrow)
- Navigation effect: can be set to *origin* (depicted as a hollow arrow end) or *destination* (filled arrow end).

Filters, defined as expressions loosely based on OCL and which are associated to links. In these expressions, a question mark (?) symbol represents user input.

All these symbols can be observed in Fig. 7. This figure depicts a possible OO-H navigation model corresponding to our checkout running example. Fig. 7 also illustrates how the process view provides the necessary information to generate a default navigation view, enabling in this way the automatic generation of a navigation model out of a process model. In order to get this navigation model, we have applied a predefined set of mapping rules, which can be summarized as follows:

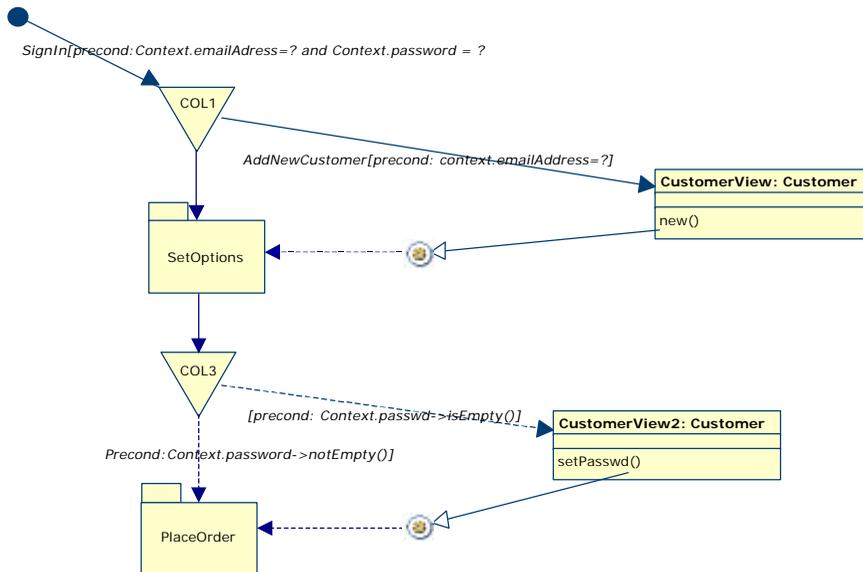
In Table 1 we observe how non-transactional activities are transformed into navigational links, which will need to be further refined with a navigation filter that completes the activity specification. Transactional activities and/or transactional subactivity states on the contrary require the support of an underlying domain operation that hides and preserves such transactional character. Operations are accessed at NAD level by means of service links. On the other hand, non-transactional subactivity states can be regarded as navigational subsystems, and therefore materialized in a OO-H Navigation target associated with each of the defined subsystems. Transitions trivially map to traversal links, which are by default activated by the user and cause a change of user view.

Activity Diagram Element	NAD diagram element
Non-Transactional Activity	Navigational link refined with precondition filter
Transactional Activity	Service link associated with a Navigational class
Transition	Navigation target
Subactivity	Service link associated with a Navigational class
Branch	Traversal link
Merge	Collection from which a set of Traversal links with exclusive filters departs.
Split-Join	Collection at which a set of Traversal links with no filters arrives.

**Table 1. Mapping Rules between Process View and Navigation View in OO-H**

Branches can be regarded as menus where only one option is available at a time. This fact is modeled in OO-H by means of a *collection* construct and a set of traversal links, each one with an exclusive filter associated. Merge constructs, on the other hand, cause the generation of a collection that is the target for a set of automatic traversal links.

Last, the synchronization bars (split-join constructs) cause the generation of a default path that traverses the concurrent activities in arbitrary order (namely from top to bottom and from left to right).



**Fig. 7. NAD Diagram for Customer Identification in the Checkout Process**

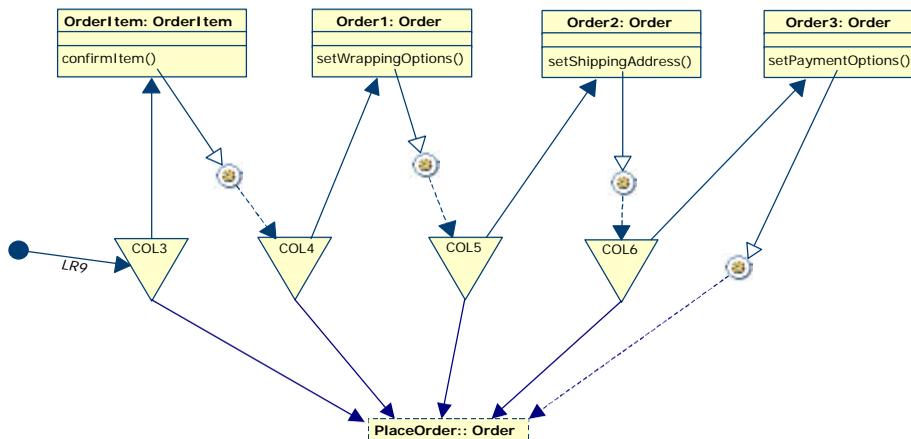
As an illustrating example, and looking back at the activity diagram of Fig. 6, we observe that the first constructor that appears is the *SignIn* non-transactional call state. This activity is materialized in Fig. 7 in a navigational link with an associate filter (OCL

formula) that implements a query over the underlying information repository. After this query has been performed, and depending on whether the user is new or a returning customer, a collection construct (*coll*, see Fig. 7) drives us either to a *new()* method or to a *SetOptions* navigation target respectively. Assuming that the user states he is new, he will follow the *AddNewCustomer* link, which first of all will demand the user to enter an *emailAddress* that is gathered in an OO-H predefined *context* object. While the customer navigational class and the associated service link have been generated automatically, the filter is a refinement added by the designer on the default model to correctly capture the Amazon interface.

This email value will be then used to provide a value to one of the parameters defined for the *new()* service that can be accessed through the *CustomerView*. When the underlying operation returns the control, and assuming that everything is OK, a system automatic traversal link (dotted arrow) drives the user to the *SetOptions* Navigation Target.

This diagram also shows the association between activities and classes and/or domain operations. As an example, the association of the *AddNewCustomer* activity of Fig. 6 with the *new()* operation in the *Customer* class has caused the inclusion of a *CustomerView* and a service link associated (see Fig. 7).

If we now enter the *SetOptions* navigation target, generated after the homonim subactivity state, we may observe how all options may be performed in parallel. Navigationally speaking, and in order to assure that the completion of all the parallel activities is possible, OO-H infers a navigation path that sequentially traverses the constructs associated with each one of these activities (see Fig. 8).



**Fig. 8. NAD Diagram Corresponding to the *SetOptions* Subactivity State**

Once the whole navigation model has been refined, a default presentation model can be automatically generated in the OO-H development environment. In order to complete the application specification, OO-H provides a presentation model that is partly-based on the design models presented so far and that falls out of the scope of this paper.

## 5. DESIGN OF WEB BUSINESS PROCESS WITH UWE

The UWE methodology [Koch & Kraus 2002] is an object-oriented and iterative approach based on the standard UML. The main focus of UWE is the systematic design followed by a semi-automatic generation of Web applications. To support the systematic design the CASE-tool ArgoUWE (an extension of ArgoUML<sup>2</sup>) is currently being implemented. The semi-automatic generation of Web applications is supported by the UWEXML – a model-driven Code Generator for deployment to an XML publishing framework. Both are part of the OpenUWE development environment. The common language for data interchange within this architecture is defined by the UWE metamodel defined as a conservative extension of the UML metamodel and therefore a MOF (Meta Objects Facility) compatible metamodel [Koch & Kraus 2003].

The UWE metamodel elements are also the basis for the UWE notation which is defined as a “lightweight” UML profile, i.e. a UML extension based on the extension mechanisms defined by UML. The UWE profile includes a set of Web specific modeling elements for navigation, presentation, process and personalization. In this section we will focus on the notation used by UWE for business processes and the development steps to build such category of applications.

The UWE design approach for Web business process, in the same way as OO-H does, is based on the models built during the analysis phase, i.e. the conceptual model and the process model, both presented in Section 4. It uses standards not only to build the analysis models, but UWE also sticks to the UML in this design phase. In this phase UWE selects the appropriate diagram types and proposes to enrich the UWE Profile with a couple of modeling elements, improving in this way the expressiveness of the UML constructs for the Web domain. In the treatment of business processes UWE differs from OO-H by not mapping the process model to the navigation model but additionally introducing specific process classes that are part of a separate process model with a clear interface to the navigation model.

Design of Web business applications following the UWE methodology requires the following activities: First, the refinement of the conceptual model adding attributes and methods to the already identified classes. We will neither detail this refinement process nor depict the resulting diagram in this work, as these are well known activities done in object-oriented development. Second, the integration of the processes in the navigation model to indicate browsing possibilities. Third, the refinement of the process model building a process structure and a process flow view. Last but not least, the presentation model is built based on the navigation and process models showing how the navigation paradigm and the business processes are combined.

### 5.1. INTEGRATION OF PROCESSES IN THE NAVIGATION MODEL

Navigation modeling activities in UWE comprise the construction of the navigation model in two steps. First, the objective is to specify *which* objects can be visited by navigation through the application. Incorporating to this diagram additional constructs it

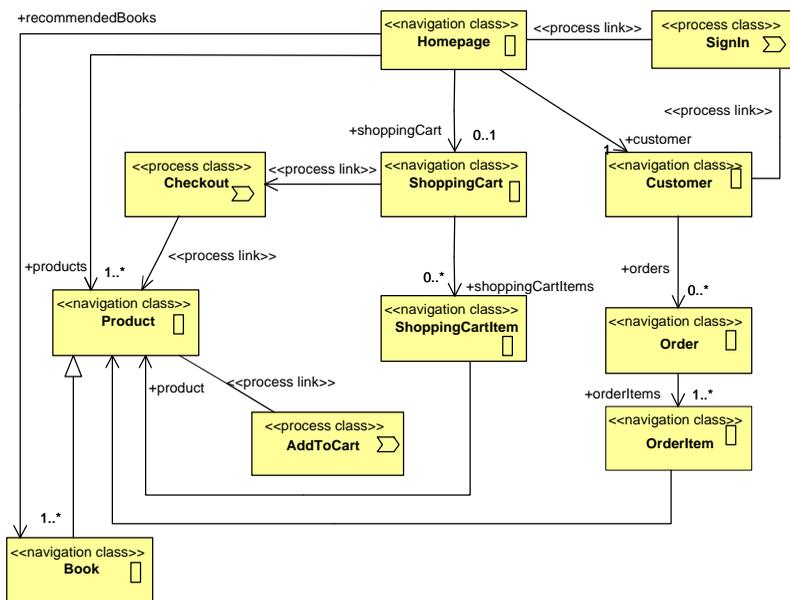
---

<sup>2</sup> [www.tigris.org](http://www.tigris.org)

is shown *how* the user can reach the navigation elements. The navigation model is represented by a stereotyped class diagram. It includes the classes of those objects which can be visited by navigation through the Web application, such as classes *Product*, *ShoppingCart*, *Order*, *Customer*, *Book*, etc. UWE provides a set of guidelines and semi-automatic mechanisms for modeling the navigation of an application, which are detailed in previous works [Koch and Kraus 2002]. This automation as well as model checking is supported by the CASE tool ArgoUWE [Zhang 2002].

UWE defines a set of modeling elements used in the construction of the navigation model. For the first step the «navigation class» and the «navigation link» have been used until now to model nodes and links. For modeling process-aware Web applications we introduce two additional stereotypes «process class» and «process link», which are defined with the following semantic:

- *process class* models a class whose instances are used by the user during execution of a process. It is possible to define a mapping function between «process class» classes and use cases (those use cases not stereotyped as «navigation») in a similar way to the mapping function defined between navigation classes and conceptual classes.
- *process link* models the association between a «navigation class» and a «process class». This process link needs to have associated information about the process state, i.e. they may be constraint by an OCL expression over the process state. This allows resuming activities within the process under certain conditions.



**Fig. 9. UWE Navigation Model (First Step) of the Checkout Process in [www.amazon.com](http://www.amazon.com)**

*Process links* in the navigation model indicate starting points of process within the navigation structure (see Fig. 9) . This process link can be bi-directional, such the case of

the process links related to *AddToCart* and *SignIn* or the model should include another «process link» that establishing where the navigation will continue after the process ends, such as by the *CheckoutProcess*. The process itself is defined in a separate model (see next section).

Fig. 9 shows the navigation model after its first construction step. Note that associations which are not explicitly stereotyped are stereotyped associations of type «navigation link» (we omit them to avoid overloading). As example of a Amazon product line we only show the «navigation class» *Book* to keep the diagram simple as no modeling differences would be shown by including other product lines, such as classes DVD or CD. Although the notation for a bidirectional link with a line without arrows is not intuitive, we prefer to stick to the UML notation.

The second step in the construction of the navigation model consists of the enhancement of the model by a set of access structures needed for the navigation. In a first step, this enhancement is partially automated, it consist in introducing indexes, guided-tours and queries. For each of these constructs UWE defines a stereotyped class «index», «query» and «guided tour». In Fig. 10 we use icons for indexes (e.g. *OrderList*) and queries (e.g. *SearchProduct*), which are defined by UWE within the UML extension mechanisms [Koch & Kraus 2001].

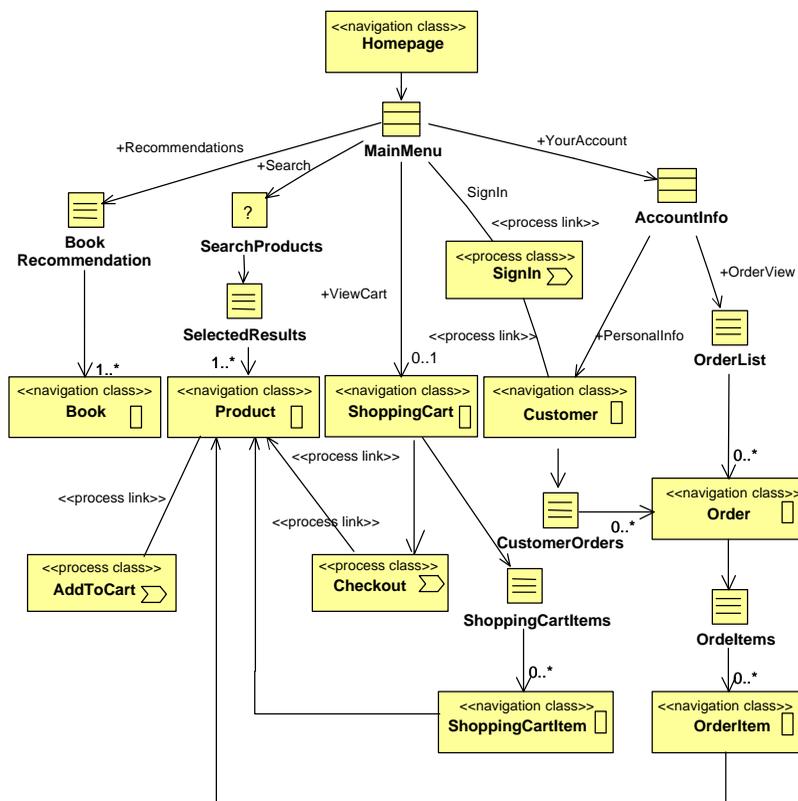


Fig. 10. UWE Navigation Model (with Access Elements) of the Checkout Process

Further the model is enriched automatically with menus, for which construct UWE includes a stereotyped class «menu». For all these constructs UWE defines the semantic based on the extension of the UML metamodel with UWE specific modeling elements and using the Object Constraint Language (OCL) to define invariants on these constructs. Fig. 10 shows the result of the complete navigation modeling process. In this second step as we use already defined UWE modeling elements, there is no need to improve this model to model Web business processes beyond the «process class» and «process link» defined above.

## 5.2. REFINEMENT OF THE PROCESS MODEL

At design level UWE proposes to build a process model which has a *structural view* and a *behavioral view*, also called the *process flow model*. Another view is the integration view with the navigation model – already presented in the previous section – which is depicted in the navigation model defining process entry and exit points between process execution and navigation. These concepts are similar to the “start activity” and “end activity” concepts of Brambilla et al. [2002]. Unlike them, however, we model the process itself independently from the navigation, emphasizing in this way the separation of aspects in the design of Web applications.

In Fig. 11 the *structural process model* for the *Checkout* process of the Amazon example is depicted. The structural process model is – like the navigation model – derived from the conceptual model. The difference to the navigation model is that the objective of this model is to capture the process related information comprising structure and behavior. As it is shown in Fig. 11 part of the process state is implicit by the cardinality 0..1 to other process classes meaning that at runtime these links exist or do not exist.

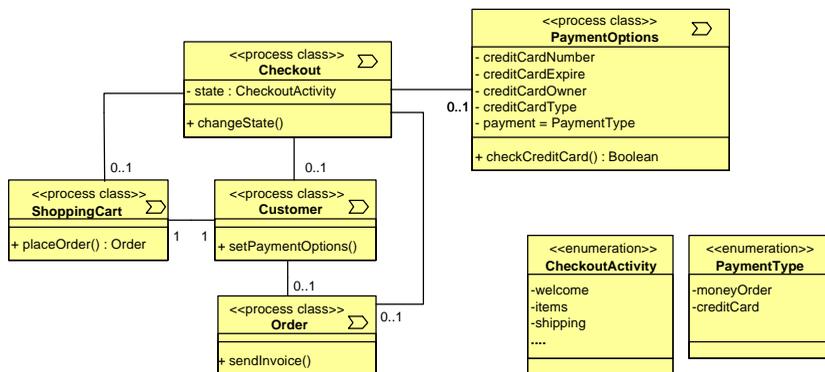
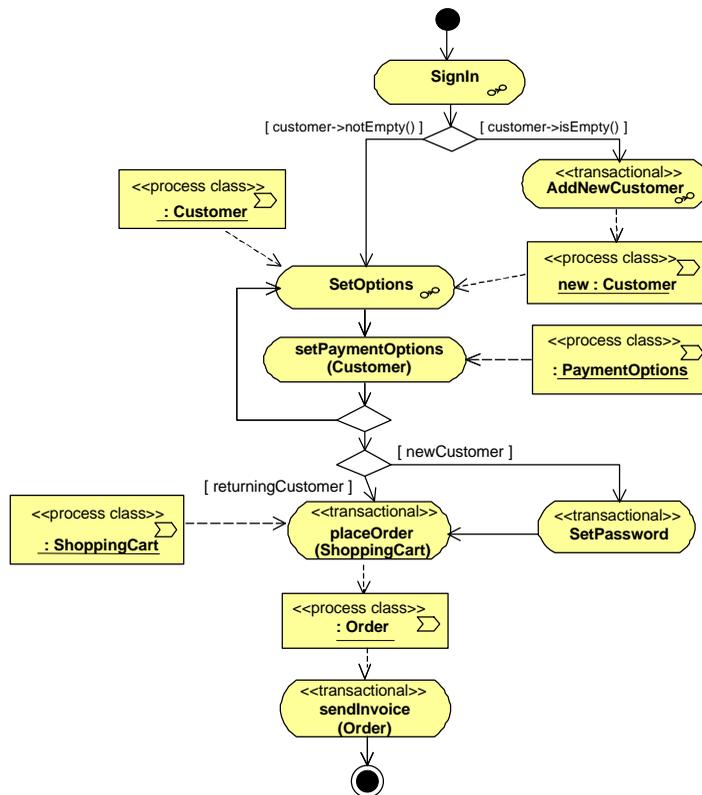


Fig. 11. Process Structural View of the Checkout Process in [www.amazon.com](http://www.amazon.com)

Conversely, we allow new modeling elements in the process model which are not derived from any conceptual model element. The notation of this model is a class diagram using the stereotype «process class». A special process class that is not derived from the conceptual model is *PaymentOptions* containing information about the payment options. The attributes of these classes express data needed by the process including user input, such as the attributes of the process class *PaymentOptions*, and process state information, such as the attribute *state* of the class *Checkout*.

Every process is assigned to exactly one process class and for all these process classes a process flow model, i.e. a UML activity diagram, is defined. The process state can be made explicit by introducing state attributes in the process class as shown in Fig. 11 or it is derived from process classes in the transitive closure concerning associations of the particular process class. Such a state allows for a re-initiation of the process after an interruption without going through all the steps the user has gone the first time. Operations are used to validate data and to change the system state in synchronization with the conceptual model. Data validation queries can be specified by OCL post conditions and are thus automatically transformable to code. For example, for the class *PaymentOptions* validation operations (*checkCreditCard*) are defined for the validation of the entered data and for the validation of the credit card information.



**Fig. 12. UWE Process Flow Model of the Checkout Process in [www.amazon.com](http://www.amazon.com)**

The *process flow model* depicted in Fig. 12 is a refinement of the process model at analysis level (see Fig. 3) consisting of UML activity diagrams. Every activity is either a UML subactivity state or a UML call state. UML defines a subactivity state as the representation of the execution of a non-atomic sequence of steps that has some duration (set of actions and possibly waiting for events). A UML call state is an action state (atomic action) that calls a single operation. Note that we strictly follow the notation and semantic that the UML defines for modeling elements used in activity diagrams, e.g.

subactivity state icon. The process flow for a subactivity state is captured in another process flow model, i.e. activity diagram. Call states can only be specified for operations; we define them for operations of process classes in the structural process model. Our example includes the call states *setPaymentOptions*, *placeOrder* and *sendInvoice*. This includes the validation of process data and the call of operations that change the process model as well as the underlying conceptual model. By supplying guard expressions on branches following such a call state we can model the process flow depending on the result of operations of the process information model. To note is that call state names are the name of the operations and are not written beginning with a capital letter.

Process class object flow states are used to express user input and output. In our example therefore, we model the call state *setPaymentOptions* explicitly (not as part of the subactivity state *SetOptions*). The *PaymentOptions* object flow state represents input from the user and the corresponding submit button in the presentation model will trigger the transition to the *setPaymentOptions* call state (see Fig. 12). Similarly to OO-H, call states may be stereotyped as «transactional» to express the transactional character of those action states.

### 5.3. SUPPORT OF PROCESSES IN THE PRESENTATION MODEL

The presentation model of UWE allows for the specification of the logical presentation of a Web application. Based on this logical model a physical presentation can be built which contains further refinements of the elements for the physical layout, e.g. font and colors. This physical representation, which is not within the scope of this work, cannot be captured by any UML model.

Within the presentation model we distinguish two different views:

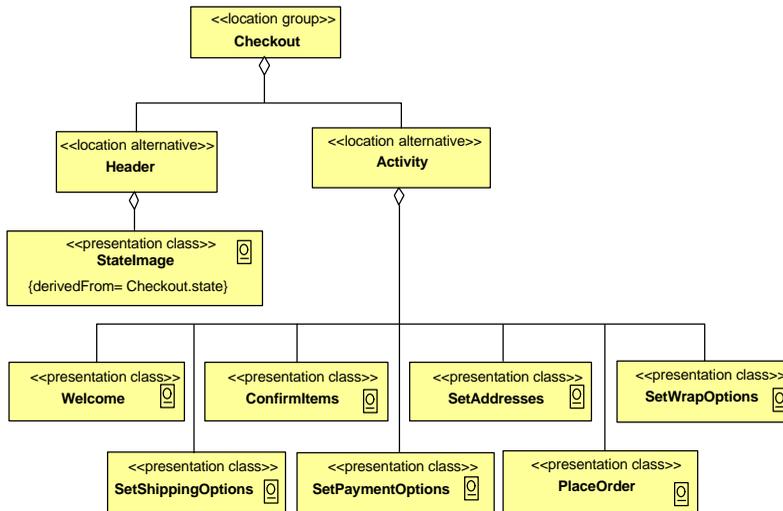
- *structural view* showing the structure of the presentation space,
- *user interface (UI) view* presenting details about the user interface elements in the pages.

The goal of the structural view of the presentation is to model how the presentation space is divided, which presentation elements are displayed in the same space (but not at the same time) and how presentation elements can be grouped. Fig. 13 shows the presentation structure view for our example the Checkout process.

The central concept around which the structuring of the presentation space takes place is the concept of location. Therefore we define in the UWE metamodel an abstract class with stereotype «location», which is the generalization of stereotyped classes we can observe at Fig. 13, i.e. «location group», «location alternative» and «presentation class». The semantic of these stereotyped classes is defined as follows:

- «location group» stereotyped classes are used to model the presentation sub-structure, e.g. as a set of pages. They aggregate a list of sub-locations.
- «location alternative» stereotyped classes are used to model presentation alternatives among «location» classes; optionally a default alternative can be specified.
- Stereotype «presentation class» represents logical page fragments and is composed of the logical user interface elements presented to the user of the application. Every «presentation class» element is related to exactly one «navigation class» element of

the navigation model or one «process class» element of the process model defining thereby the presentation for this particular element.



**Fig. 13. Presentation Structure View of the Checkout Process**

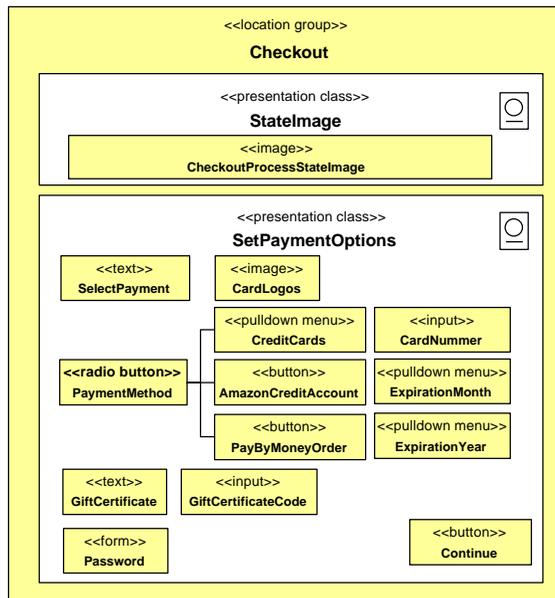
As shown in Fig. 13 the location *Checkout* is divided in two alternative sub-locations exactly one *Header* and one alternative location *Activity* whereas the activities of the checkout process are presented for the interaction with the user. The Header includes an image *StateImage* which visualizes the current step (activity) of the checkout process. The dependency to the current activity is modeled by the tagged *derivedFrom* with value given by the attribute *state* of class *CheckoutProcess*.

Fig. 14 depicts the detailed *user interface view* of the presentation class for the *PaymentOptions* process class. We use an alternative UML notation for the composition relationship showing the composed user interface (UI) elements within the visual container of a presentation class. Although this notation is not supported by most of the UML CASE Tools, we use it in this work as it allows for a more intuitive sketch as the traditional composition relationship when depicting *the user interface view*.

The presentation class *SetPaymentOptions* is presented as part of the location group *Checkout* together with the presentation class *SetImage* that shows in an image the current state of the checkout process. Every type of user interface element has a stereotype associated with it, e.g. «text», «image», «radio button», etc. They are connected to the features (i.e. attributes or operations) of the underlying navigation or process classes in the case of dynamic user interface element. Additionally, these types of user interface elements can be used for static user interface elements as in the example static text (*SelectPayment*) or static images (*CardLogos*).

The type of user interface element used to present the corresponding elements of the underlying models depends from their type and the intended use. An «input» element for example can be used for displaying information as well as for information input in the case of attributes of a process class, e.g. the *CardNumber* element in the example. The

«radio button» element can be used to express the choice between different alternatives, such as the payment methods in our example. The attached user interface elements reflect the active UI elements for each case.



**Fig. 14. UI Elements View for the Presentation Class *SetPaymentOptions***

A special case is the «button» element *Continue* (see Fig. 14) which triggers the *setPaymentOptions* call state in the corresponding process model.

## 6. CONCLUSIONS

The inclusion of process definition mechanisms in the context of Web methodologies is not only a must, imposed by enterprise demands, but also convenient from the point of view of increased support to the application evolution. This evolution support is necessary due to the frequent appearance of new or changed business requirements. In this sense, we believe that the greater flexibility that comes with the explicit definition of such processes will induce a faster implementation of these changes.

Being the notation associated with Web Engineering methods and methodologies so different from approach to approach, the first temptation is to strive (as we have been doing up to now) to find individual solutions to this new challenge. This effort, enriching as it is, suffers from the danger of providing enterprises and researchers with different vocabularies, constructs and models to refer to eventually very similar concepts (although usually with different nuances). Such little differences are however enough as to make very difficult for us to reach general agreements, as different research events had shown in the past.

That is the main reason why in this article we have tried to work the other way round; OO-H and UWE are very different proposals. On one hand the OO-H method follows a bottom-up approach, uses standards only in some phases of the modeling process and tries to keep the set of diagrams to a minimum in order to ease the work of model compilers for the automatic generation of Web interfaces.

On the contrary, UWE is exclusively based on standards. It is a top-down approach that defines their modeling elements based on a metamodel defined in UML and defined as an extension of the UML metamodel. UWE uses whenever possible the constructs provided by the UML and in some cases extends the notation to support the Web development specific characteristics. The extensions are then strictly performed according to the extension mechanisms provided by the UML. In addition, UWE focus on a systematic development process, but this subject was not within the scope of this work.

In spite of these differences, it was possible to reach agreements on the main concepts to be included in both, OO-H and UWE, and to define a common approach for the modeling activities during the analysis phase of Web business intensive applications. Among these agreements, it is interesting to note how both approaches opted, unlike previously existing proposals [Bambrilla et al. 2002] for defining a separate model to address process concerns. We believe that providing separate models not only eases the construction and maintenance of such models, but also reflects the fact that the same process may be the basis on which different interfaces may be defined, all of them giving support to this process.

Another important contribution of this work is the identification of at least two possibilities for the treatment of process concerns in the design phase of Web applications development. On one hand, OO-H has opted for the definition of default mapping rules that make possible the definition of default navigation maps based on the defined activity flows. OO-H therefore considers that the purpose of certain navigation links may be regarded as that of guiding the user through the different process steps. OO-H comes together with a prototyping environment that is based on its navigation diagram. Embedding process concerns in this navigation diagram makes trivial the prototyping of such process in order for the user to validate it. Furthermore, in this way we have kept the set of design constructs needed to define a Web interface to a minimum.

UWE instead has opted for design flows of control for process modeling in addition to the navigation model, which is only enriched to reflect a set of integration points, that is, points in which the user may leave the navigation view to enter a process view. At presentation level the same set of presentation modeling elements is used to support both, the navigation and the process. This loose integration supports a clear separation of concerns and enables reuse of processes, such as customer login and checkout, in different context or applications. Furthermore, it eases the maintenance and Web application evolution.

## **7. FUTURE WORK**

We plan to include the extension to our methods OO-H and UWE to support business process modeling in our CASE tools, VisualWADE and ArgoUWE, respectively.

VisualWADE needs to include support both for UML activity diagrams and for the new «transactional» stereotype. Also, we are working on the refinement of the mapping process between activity diagrams and NAD. In this sense, the definition of OCL guard conditions associated with transitions may provide automatic generation of some of the filters included at NAD level. Also, detailed object flows complementing the activity diagram may simplify the definition of the service interfaces affected.

ArgoUWE will be extended to support process modeling as defined in this work. The new modeling elements have been already included in the UWE Metamodel. The consistency between the process model and the already existing navigation and presentation models will be checked on the basis of OCL constraints that improve the already existing set of about 20 constraints used for the UWE model checking.

## REFERENCES

- [Atzeni & Parente 2001] Paolo Atzeni, Alessio Parente (2001). Specification of Web Applications with ADM-2. 1st International Workshop on Object Oriented Software Technology. Valencia, Spain.
- [Baresi et al. 2001] Luciano Baresi, Franca Garzotto, Paolo Paolini.(2001). Extending UML for Modeling Web Applications, 34th Hawaii International Conference on Systems Sciences.
- [Brambilla et al. 2002] Marco Brambilla, Stefano Ceri, Sara Comai, Piero Fraternali. (2002). Specification and Design of Workflow-Driven Hypertext. Journal of Web Engineering, Vol. 1, No. 1.
- [Cachero 2003] Cristina Cachero. (2003). OO-H: Una extensión a los métodos OO para el modelado y generación automática de interfaces hipermediales. Available online at <http://www.dlsi.ua.es/~ccachero/pTesis.htm>
- [Cachero & Gomez 2002] Cristina Cachero Jaime Gómez. (2002). Advanced Conceptual Modeling of Web Applications: Embedding Operation Interfaces in Navigation Design. 21th International Conference on Conceptual Modeling. El Escorial, Madrid. Nov. 2002.
- [Ceri et al. 2002] Stefano Ceri, Piero Fraternali, Mariestella Matera. (2002). Conceptual Modeling of Data-intensive Web Applications. IEEE Internet Computing 6 (4): 20-30.
- De Troyer & Casteleyn 2001] Olga de Troyer, Sven Casteleyn. (2001). The Conference Review System with WSDM. 1st International Workshop on Object Oriented Software Technology. Valencia, Spain.
- [Escalona & Koch 2003] María José Escalona, Nora Koch (2003), Ingeniería de Requisitos en Aplicaciones para la Web: Un Estudio Comparativo, IDEAS'03.
- [Gomez et al. 2001] Jaime Gómez, Cristina. Cachero, Oscar Pastor. (2001). On Conceptual Modeling of Device-Independent Web Applications: Towards a Web Engineering Approach. IEEE Multimedia 8(2): 20-32. Special Issue on Web Engineering.

- [Jacobson et al. 1992] Ivar Jacobson, Magnus Christersen, Patrik Jonsson, Gunner Overgaars. (1992). Object-oriented Software Engineering: A Use Case Driven Approach, Addison Wesley.
- [Kahn et al. 1957] Robert Kahn, Charles Cannell. (1957). The dynamics of interviewing; theory, technique, and cases, New York, Wiley.
- [Koch & Kraus 2002] Nora Koch , Andreas Kraus. (2003), The expressive power of UML-based engineering, Proceedings of the IWWOST'02, CYTED, 105-119.
- [Koch & Kraus 2003] Nora Koch, Andreas Kraus (2003), Towards a Common Metamodel for the Development of Web Applications, International Conference of Web Engineering, LNCS, Springer Verlag, to appear.
- [Korthaus 1998] Axel Korthaus (1998). Using UML for Business Object Based Systems Modeling. UML Workshop 1997. Mannheim, Germany.
- [Markopoulos 2000] Panos Markopoulos. (2000). Supporting Interaction Design with UML, Task Modelling , TUPIS Workshop at the UML'2000.
- [Nunes & Cunha 2000] Nuno Nunes, José Cunha. (2000). Towards a UML Profile for Interaction Design: The Wisdom approach. Proceedings of the Unified Modeling Language Conference, UML'2000, Evans A. and Kent S. (Eds.). LNCS 1939, Springer Publishing Company, 100-116.
- [Retschitzegger & Schwinger 2000] Werner Retschitzegger, Wieland Schwinger. (2000). Towards Modeling of Data Web Applications - A Requirement's Perspective. American Conference on Information Systems AMCIS 2000, Vol. 1, 149–155.
- [Schwabe et al. 2001] Daniel Schwabe, Esmeraldo, L., Gustavo Rossi, Fernando Lyardet. (2001). Engineering Web Applications for Reuse. IEEE Multimedia. Special Issue on Web Engineering, 01-03, 20–31.
- [UML 2003] UML 1.5 Standard, OMG (2003). [www.omg.org](http://www.omg.org)
- [Zhang 2002] Gefei Zhang (2002). ArgoUWE: a CASE Tool for Web Applications. [www.pst.informatik.uni-muenchen.de/projekte/argouwe](http://www.pst.informatik.uni-muenchen.de/projekte/argouwe)