Journal of Web Engineering, Vol. 3, No.1 (2004) 022-049

INTEGRATION OF BUSINESS PROCESSES IN WEB APPLICATION MODELS

NORA KOCH AND ANDREAS KRAUS

Ludwig-Maximilians-Universität München, Germany
{kochn,krausa}@informatik.uni-muenchen.de

CRISTINA CACHERO AND SANTIAGO MELIÁ

Universidad de Alicante, Spain
{ccachero,santi}@dlsi.ua.es

Received October 15, 2003
Revised February 19, 2004

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 modeling support provided for this kind of processes by the Object-Oriented Hypermedia method (OO-H) and the UML-based Web Engineering (UWE) approach. Both methods apply UML use cases and activity diagrams, and supply UML standard 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 www.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 for business processes.

Key words: Web Engineering, Business Process, Object-oriented Design Method, UML, Visual Modeling, Process Modeling, UML Profile
Communicated by: D Schwabe and P Fraternali

1 Introduction

Business processes, regarded as heavy-weighted flows of control consisting of activities and transitions [23], have always played an important role in software development methods, to the point that many proposals include the definition of an explicit view (the *business process view*) in order to address their complexity. In contrast, such processes have been only tangentially tackled in most existing Web Application modeling approaches [20]. This is partly due to the fact that most of these methods were born with the aim of successfully modeling Web Information Systems (IS) [2, 7, 8, 11, 15, 22], whose main focus is to store, retrieve, transform and present information to the users. This conception is however changing: requirements posed on modern Web applications have caused them no longer to be regarded as Information Systems but as Business Systems, that is, applications centered on goals, resources, rules and business processes.

In order to address these new concerns, the Web Engineering community has been working for some time on the extension of Web modeling methods with new mechanisms that permit the definition of "lightweight" business processes. In this way Web methodologies are now able to model process

aware Web applications that, based on an underlying workflow model, support and guide the user through these processes while preserving the hypertext flexibility. Such interface guidance, largely dismissed when developing traditional software systems (including Workflow Management Systems) is from our point of view a main contribution of the Web Engineering community to the field.

The extension to cover business processes in Web applications may be done following at least two different approaches: on the one hand, traditional content, navigation and/or presentation models may be enriched to capture business workflows, such as in WebML [4] and OOHDM [21]. 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 line jointly followed by the UML-based Web Engineering approach (UWE) and the Object-oriented Hypermedia Method (OO-H), as we will present in this paper. A deeper discussion of the differentiating characteristics of some of these and other approaches will be outlined in Section 7.

UWE [15] is a UML-based software development approach with the main focus on the systematic design followed by a semi-automatic generation of Web applications. UWE uses standard UML notation whenever possible and defines a "lightweight" UML profile to model the Web specific aspects. In this way UWE models for Web applications can be produced by any UML CASE tool. The systematic design and model checking is specifically supported by ArgoUWE[1] [14] an extension of the open-source CASE tool ArgoUML[2]. The automatic generation is provided by the development environment OpenUWE[3] that is currently being implemented.

OO-H [5] is a Web interface development approach that fosters the use of UML-compliant analysis models and a set of proprietary design models and constructs, specially suited to speed up the development process for the definition of Web interfaces. The method also provides mechanisms to establish integration points between the resulting interface models and legacy modules. The set of analysis and design steps proposed by OO-H are supported by a modeling environment called VisualWADE[4]. Additionally, VisualWADE provides a set of model compilers in order to generate a running application for several different platforms and languages.

Our aim – described in this paper – has been to propose a common set of modeling concepts that suffices to define sound, non-trivial business processes in the methods OO-H and UWE, 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 [17, 18] have already acknowledged its feasibility and excellence for non-Web applications. From the set of modeling techniques provided by the UML, the activity diagram, which is the most suitable mechanism to model the business workflow, has been adopted to define the different processes at analysis level [24]. In activity diagrams, 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 addition, in this article we deal with the integration of business processes into the hypertext structure when modeling process-aware Web applications. Regarding this topic, in this paper we present two slightly different solutions implemented by OO-H and UWE, respectively. UWE

[1] http://www.pst.informatik.uni-muenchen.de/projekte/argouwe
[2] http://www.tigris.org
[3] http://www.pst.informatik.uni-muenchen.de/projekte/openuwe
[4] http://www.visualwade.com

maintains the *separation of concerns* at design level by providing separate models for process and navigation concerns. OO-H, on the contrary, preserves a single navigation model that, now, assures the fulfillment of the process restrictions, offering an *integrated design view* of both aspects.

This work is organized as follows: Sections 2 discusses how to model business processes in requirement analysis of Web applications. Section 3 presents the models to be built during analysis for this kind of application, a common solution for both methodologies. Section 4 and Section 5 describe the design steps for OO-H and UWE, respectively. Section 6 summarizes the main lessons learnt from the experience of working together on a common solution for two approaches as different as OO-H and UWE. Section 7 presents an overview of related work in the field of modeling Web business processes. Last, Section 8 outlines the conclusions while Section 9 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 modeling of process-aware Web applications 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 include 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 [10] 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" [13], and use case modeling is a well known formalism for graphical representation and description of requirements of business intensive – Web or non-Web – applications that has been integrated in the UML [12].

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. Use case models include 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 – action sequences. In order to address this additional complexity (as it is shown in the next section) both approaches propose the supplementary use of UML activity diagrams.

In our Amazon running example we have identified two actors that play a relevant role in the checkout process: the *NonRegisteredUser* and the registered user, named C*ustomer*. The non-registered user can – among other activities – search and select products, add products to the shopping cart and get registered. The *Customer* inherits from the *NonRegisteredUser* and is allowed to start the checkout process after signing in with a valid ID and password.

Figure 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 Amazon 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.

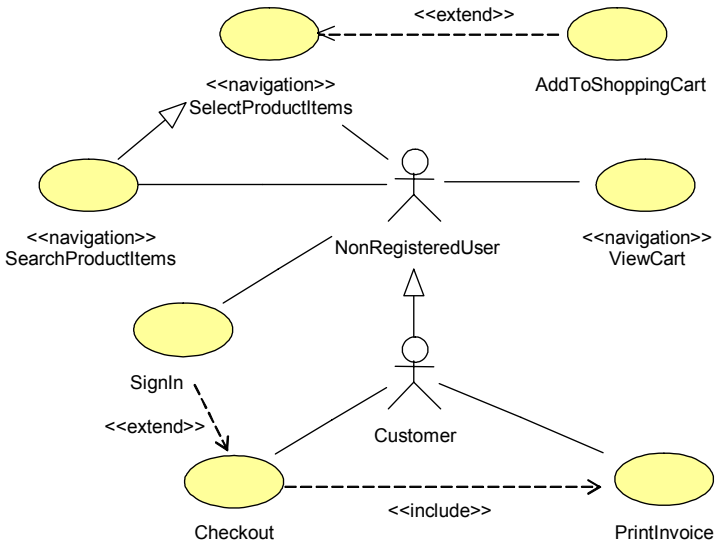

Figure 1. Use Case Diagram of the Checkout Process in www.amazon.com

At any time, the user may decide to check the items included so far in his shopping basket (use case *ViewCart*). Also, he could decide to personalize his view, for what he would have to sign in. 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 *PrintInvoice* 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 Figure 1 this fact is captured by means of an additional association between the actor *Customer* and the *PrintInvoice* 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 as they only express navigation activities. For this kind of use cases, we propose the use of a «navigation» stereotype, as defined in [2]. 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, the next step consists in the analysis of the problem domain. This analysis phase has traditionally involved in both methods – OO-H and UWE – 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 Figure 2.

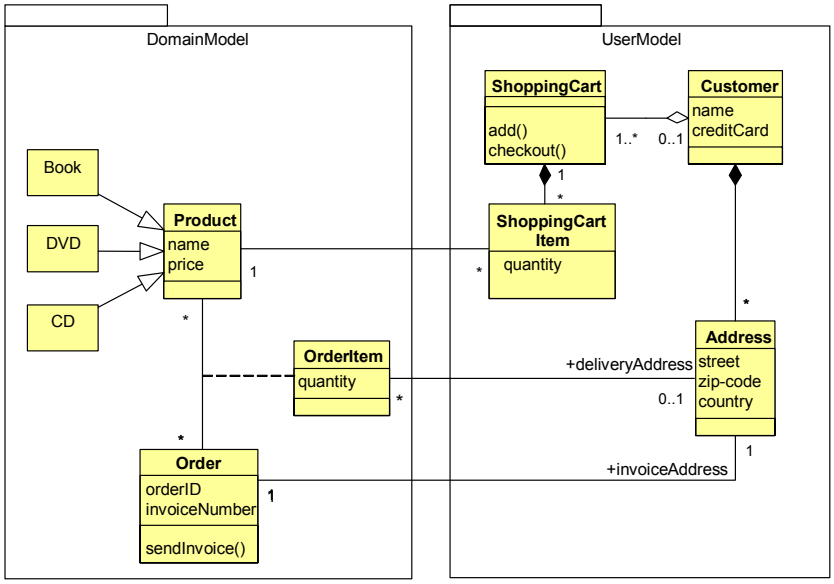

Figure 2. Conceptual Model of the Checkout Process in www.amazon.com

This class diagram is made up of two packages: the User Model and the Domain 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 Domain 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 signing-in in the system) has a *ShoppingCart*, which is made-up of S*hoppingCartItems* (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 *ShoppingCartItems* into *OrderItems*. 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 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 [18] is a UML extension that proposes the use of a set of stereotyped classes that make the notation not very intuitive. Markopoulos [17] instead makes two different proposals: a 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) either to a UML classifier, such as a use case, or to a package, or to the implementation of an operation [23]. 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. Figure 3 presents an activity diagram that shows the simplified flow of control of the Amazon Checkout process (depicted as a non-navigational use case in Figure 1).

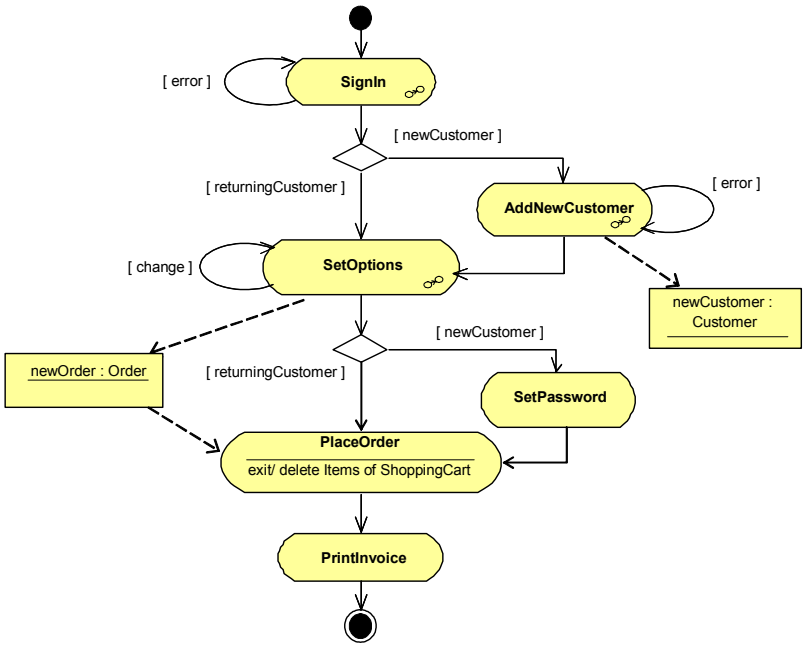

Figure 3. Process Model of the Checkout Process in www.amazon.com

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 either be added to the system or otherwise directly driven to the *SetOptions* subactivity state that permits the user to establish every purchase option (see

Figure 4). Once this activity 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 Figure 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, Figure 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 the confirmation of 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. We would like to note that, although not shown in the example, activity diagrams provide *swimlanes* to model which actor performs each action, allowing in this way the definition of not only single but also multi-user processes in a seamless way.

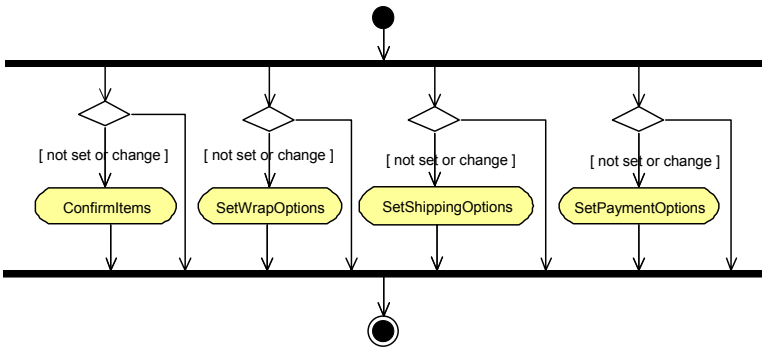

Figure 4. Activity Diagram of the SetOptions Process in www.amazon.com

In Figure 3 we can also observe how the *use* and *extend* dependencies defined in the use case diagram of Figure 1 influence the flow of control of the process, and are materialized in the inclusion of complementary activities (see e.g. *PrintInvoice)* 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 (see Fig. 3) a *new Customer* is created as result of the *AddNewCustomer* activity and a new *Order* object is created as a result of the *SetOptions* 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 defined in the process workflow.
- 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 has opted for the first approach, while UWE follows the second one. It is important to stress that, in spite of this fact, the analysis models presented so far, are fully reusable and a common ground for discussion.

4 Design of Web Business Processes with OO-H

OO-H [11] is a generic approach, partially based on the object-oriented paradigm, which 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 development environment that includes a set of model compilers to provide automatic code generation capabilities.

OO-H bases the design process model on the concept of service, regarded as an interface whose purpose is to provide a way to partition and characterize groups of operations [23].

Services in OO-H can be classified according to several orthogonal characteristics, namely (1) synchronicity, (2) activating agent, (3) granularity (number of operations supporting the service) and (4) transactionality [6]. Inside this group, OO-H has for long been well suited to provide an arbitrary complex interface to services that are synchronous, user-activated and *uni-granular*, where by *uni-granular* we mean services supported by exactly one underlying domain operation. That is the case of Create, Delete and Update operations, typical uni-granular, transactional services which are at the core of most Web Applications.

On the other hand, processes can be regarded, from the user interface point of view, as multi-granular (compound services), that is, services that involve more than one domain operation. The definition of an interface for this kind of services presents special challenges; in these services, the user interface is responsible not only for invoking each domain operation but also for guiding the user through them following a predefined flow of control, which may involve both activity and information dependencies. The *checkout* process included in Amazon is, from the OO-H point of view, a compound, non-transactional service. With the addition of activity diagrams representing the application flow of control such as the one presented in Figure 3, the expressive power of OO-H is therefore increased to allow the modeling of not only uni-granular but also multi-granular services.

The design of Web business processes following the OO-H method consists of four steps. First, the designer must refine the conceptual model (addition of properties and even new elements to the UML class diagram). Second, he must also refine the analysis process model. During this refinement, the link between the process design constructs and the conceptual model is established by the identification of UML call states (atomic actions that call a single operation)[5] that call single domain operations and, if necessary, also by the use of domain attributes and operations both in the branch conditions and in the onEntry/onExit/Do sections of each activity/subactivity state. After this refinement, the designer must apply a set of predefined mapping rules that permit the automatic generation of a default Navigation Access Diagram based on the design process model, and which assures the traceability between both models. On this default diagram, the designer may again perform

[5] At the time of writing this article it seems that the term *call state* will be substituted in the UML 2.0 specification (probably becoming a standard in summer 2004) by a set of *call action types*, from which we would use *call operation actions* instead of *call states.*

any desired refinement, such as the addition of filters or extra navigation paths. Last, once the navigation model is completed, he can proceed to define the presentation view. Next we are diving into each one of these steps.

4.1 Conceptual Model Refinement

The first step to refine the analysis process view is to further detail the conceptual class properties. A new class diagram corresponding to our Amazon example, with a more exhaustive list of attributes and methods is shown in Figure 5. Note how this diagram is an evolution of the one presented in Figure 2.

4.2 Process Model Refinement

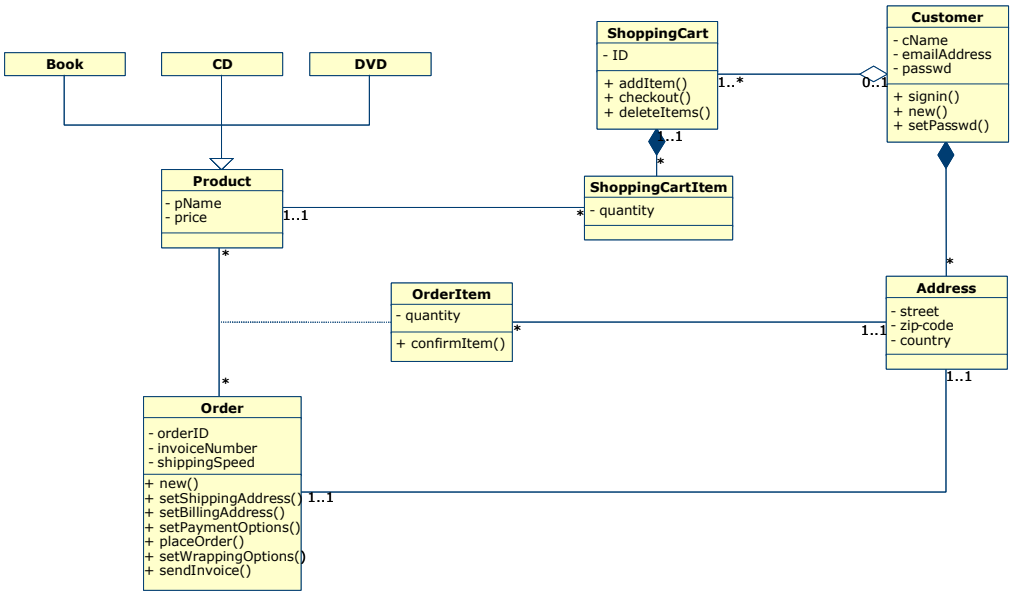

Figure 5. OO-H Refined Class Diagram of the Checkout Process in www.amazon.com

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

- Some subactivity states from the analysis model may be redefined as call states, implying that either a single operation or a navigation path through a set of data items recovered by an underlying query gives them support. That is the case of the *SignIn* or the *SetPasswd* subactivity states (see Figure 3) that, being supported by the *signIn()* and *setPasswd()* operations respectively, have been now redefined as call states (see Figure 6).
- Some call states may be merged under a common subactivity state in order to model any extra feature that involves all the grouped call states. This feature is especially useful when a transaction is detected which involves several call states. In our example, we have considered that *PlaceOrder* and *PrintInvoice* 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 Figure 6).

- A new «transactional» stereotype may 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 (atomicity, consistency, isolation and durability). 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, in Figure 6 not only the PlaceOrder subactivity state but also the *AddNewCustomer* call state - supported by the *Customer.new()* operation - has been defined as transactional, meaning that it requires underlying business logic support to guarantee the ACID properties.

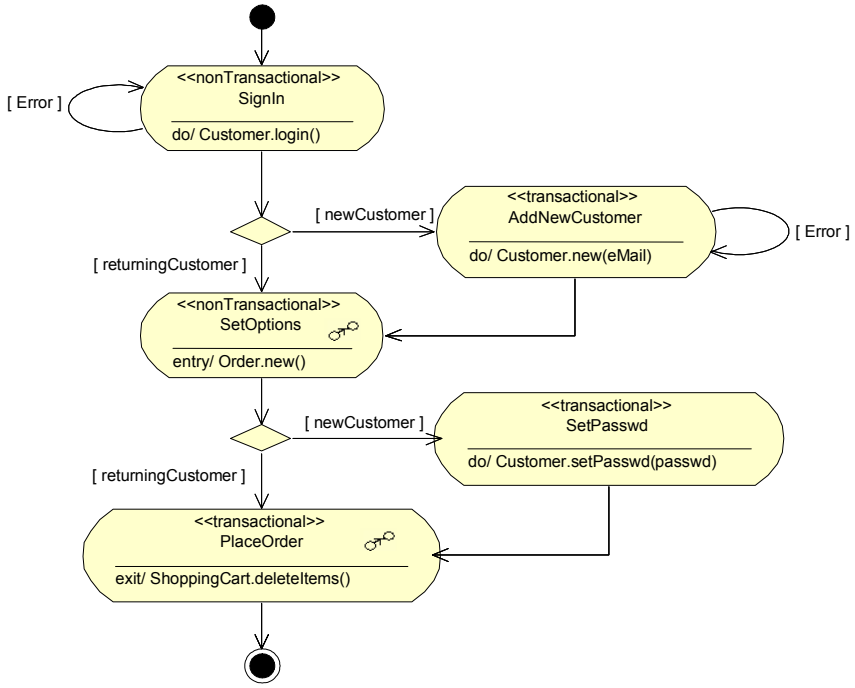

Figure 6. OO-H Refined Activity Diagram of the Checkout Process in www.amazon.com

For the sake of simplicity, in Figure 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 with 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.

4.2 Default Navigation Model

The navigation model in OO-H is defined by means of a Navigation Access Diagram (NAD). This diagram is made up of collections (menus, depicted as an inverted triangle), navigation targets

(navigation subsystems, depicted with a package symbol), navigation classes (views on conceptual classes, depicted with the class symbol) and navigation links (paths the user may follow through the system, depicted with the arrow symbol). Navigational links, being the core construct 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, (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: it can be set to *user* (depicted as a solid arrow) or *system* (depicted as a dotted arrow)
- Navigation effect: it 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 Figure 7. This figure depicts the OO-H default navigation model corresponding to our checkout running example. This default navigation view is derived from the process view presented in Figure 6 by applying the set of mapping rules that are summarized in Table 1.

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	Traversal link
Subactivity	Navigation Target
Branch	Collection from which a set of Traversal links with exclusive filters departs
Merge	Collection at which a set of Traversal links with no filters arrives.
Split-Join	Default path that traverses the concurrent activities sequentially from left to right

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

In this table 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. Branches naturally map to OO-H *collection* constructs and a set of traversal links, each one with an exclusive filter associated. Merge constructs, on the other hand, they 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).

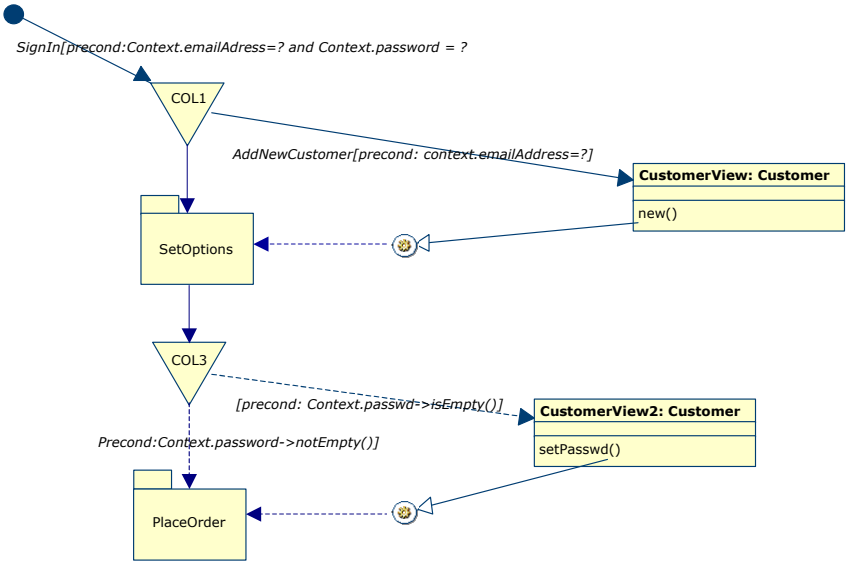

Figure 7. NAD Diagram for Customer Identification in the Checkout Process

Looking back at the activity diagram of Figure 6, we observe that the first element that appears is the *SignIn* non-transactional call state. This activity is materialized in Figure 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 (*col1,* see Figure 7) drives us either to a *new()* method or to a *SetOptions* navigation target respectively. Assuming that the user states that he is new, he will follow the *AddNewCustomer* link, which first of all will demand the user to enter an *emailAddress*. While the customer navigational class and the associated service link have been generated automatically, the filter that allows to model this user interaction 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 Figure 6 with the *new()* operation in the Customer class has caused the inclusion of a *CustomerView* and a service link associated (see Figure 7).

If we now explode the S*etOptions* navigation target, generated after the homonym subactivity state (see Figure 6), 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 Figure 8).

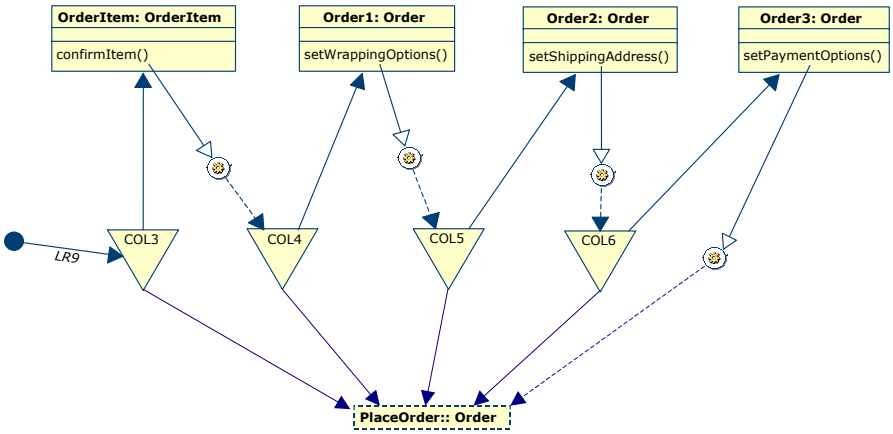

Figure 8. NAD Diagram Corresponding to the *SetOptions* Subactivity State

As we have stated before, default navigation models can be enriched not only with additional filters but also with additional navigation paths that flexibilize the user navigation experience. These additional paths may pose some risks when they imply to step out of a path that drives the user through a process. In order to control this risk, two solutions are possible. On one hand, it is possible to ban the definition of such additional navigation between process-derived navigational constructs and non-process navigational constructs. On the other hand, as OO-H suggests, it is possible to define, associated to each process, a UML statechart diagram with three states (*active*, *suspended* and *cancelled*) and whose events (associated to transitions among these three states) correspond to *navigation events,* that is, to the user activation of certain navigation links. Additionally, a history state indicator on such states reflects the need to save and/or retrieve certain information as a result of such navigation events. Back to our Amazon example, its constriction to a single process and the provision of a single navigation path that strictly follows the process steps makes the construction of such additional model unnecessary.

4.4. The OO-H Presentation View

Over the defined navigation model, OO-H provides the mechanisms to automatically construct a presentation model compliant with such navigation specification. OO-H keeps the information related to the navigation through processes at navigation level, and relies on presentation only to depict the characteristics of each navigation path. As the reader may already have inferred, links automatically generated from process models keep their process-property, and so presentation may make use of such property to change certain visibility features: different colors, explicit specification of the path that the different links belonging to a same process make up, and so on. Interested readers on the presentation capabilities of OO-H are referred to [6].

5 Design of Web Business Processes with UWE

The UWE methodology [15] is an object-oriented and iterative approach based on the standard UML [23]. To restrict notation and diagrams to those provided by the UML has the important advantage to make use of all benefits and tools that support UML. The main focus of UWE is the systematic design followed by a semi-automatic generation of Web applications. The CASE tool ArgoUWE (an extension of ArgoUML) was developed to support the systematic design [14]. The semi-automatic generation of Web applications will be supported by the UWEXML tool – a model-driven Code

Generator for deployment to an XML publishing framework that is currently being implemented. Both are part of the OpenUWE development environment that will support the complete life cycle for the development of Web applications. The common language for data interchange within this architecture is specified by the UWE metamodel defined as a conservative extension of the UML metamodel and therefore a MOF (Meta Objects Facility) compatible metamodel [16].

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 the UML (stereotypes, tagged values and OCL constraints). 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 steps to develop (i.e. the systematic design of) such type of applications.

The UWE design approach for process-aware Web applications, 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 the 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 not only in the notation used, but also by additionally introducing specific process classes that are part of a separate process model with a clear interface to the navigation model instead of mapping the process model 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 and already outlined in Section 4.1. Second, the integration of the processes interfaces 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 at the presentation level.

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. By incorporating to this diagram additional constructs it 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 the 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 [15]. This automation as well as model checking is supported by the CASE tool ArgoUWE [14].

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.

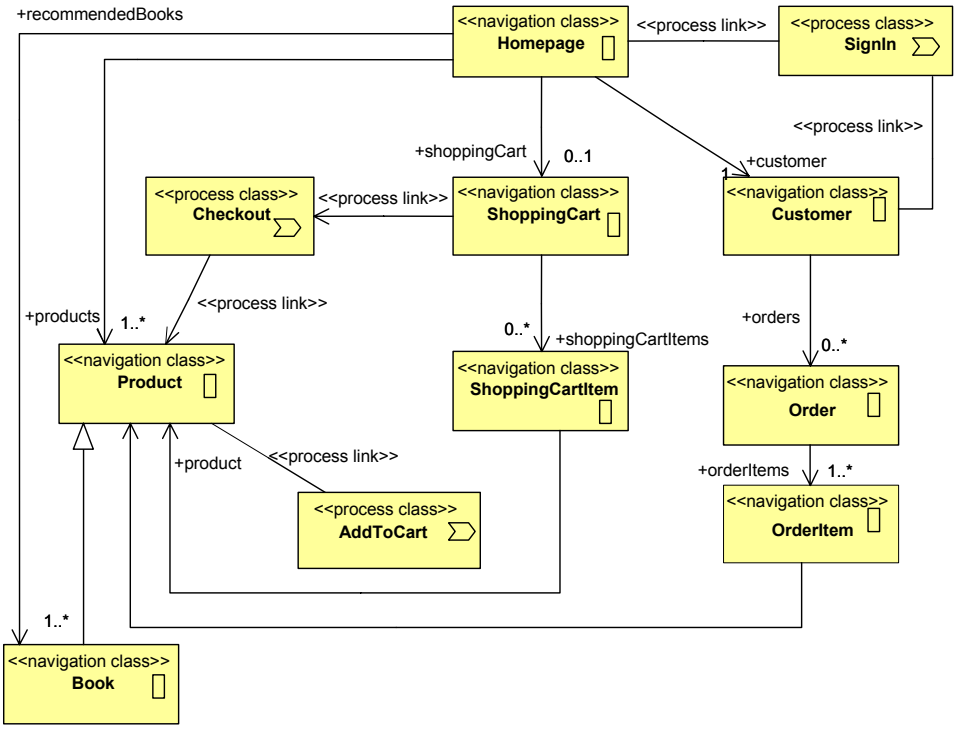

Figure 9. UWE Navigation Model of the Checkout Process in www.amazon.com (First Step)

UWE defines the following rules supporting a systematic enrichment of the navigation model with process modeling elements:

- Define a process class for each use case not stereotyped as «navigation» and not related through an «include» to another use case.
- Define at least one uni-directional association of type «process link» between each «process class» and a «navigation class». Such a «process link» indicates which «navigation class» is a starting point for the process.
- Define one uni-directional «process link» between each «process class» and a «navigation class» indicating where navigation will continue after the process ends.
- A bi-directional «process link» may replace two uni-directional associations with same source and same target classes.

Figure 9 depicts together with the navigational classes the process classes derived from the use cases *AddToShoppingCart, SignIn* and *Checkout*, i.e. *AddToCart* (with a slightly different name), *SignIn* and *Checkout.* Figure 9 shows bi-directional process links, such as in the case of those related to process classes *AddToCart* and *SignIn* or an uni-directional process link, such as by the class *CheckoutProcess*. The workflow of the process itself is defined in a separate model (see next section). Note that associations in Figure 9 which are not explicitly stereotyped are stereotyped associations of type «navigation link» (we omit them to avoid overloading). As example of the 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 the 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. This enhancement is partially automated, it consist in introducing indexes, guided-tours and queries following certain rules, which can be summarized as follows [15]:

- Introduce automatically an access element of type index whenever in the navigation model there is an association with multiplicity greater than one at the destination end, i.e. on the directed association end. Role names are moved from the destination end to the association end reaching the index.
- Introduce manually queries before indexes whenever necessary, e.g. when the number of elements of an index would require scrolling or whenever wanted.
- Replace manually indexes by guided-tours if preferred.

For each of these constructs UWE defines a stereotyped class «index», «query» and «guided tour». In Figure 10 we use icons for indexes (e.g. *OrderList)* and queries (e.g. *SearchProduct*), which are defined by UWE within the UML extension mechanisms [3].

Further the model is enriched automatically with menus, for which UWE includes a stereotyped class «menu». The rules applied to enhance the model with menus, can be summarized as follows:

- Introduce automatically a class menu for all associations that have as their source a navigation class, which has at least one outgoing association. The association between a navigation class and its corresponding menu class is of type composition. The menu includes a menu item for each role at the end of the directed outgoing associations.
- Reorganize manually a menu in a menu with submenus, if it is required by the number of menu items.

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. Figure 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.

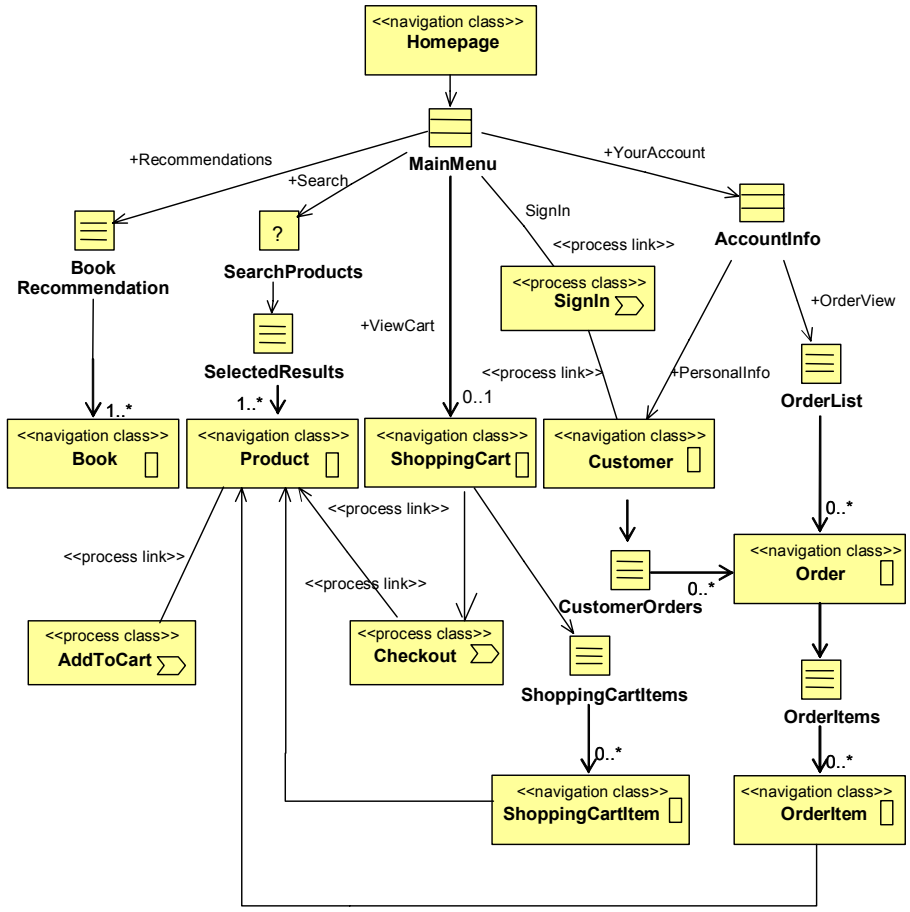

Figure 10. UWE Navigation Model (with Access Elements) of the Checkout Process (Second Step)

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 [4]. 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 Figure 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 developer is who determines the conceptual classes which are relevant for the workflow of the process, in the same way he indicates which are the navigation relevant classes. 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 Figure 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.

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, such as credit card number or credit card owner which the user only can enter during the process. 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*.

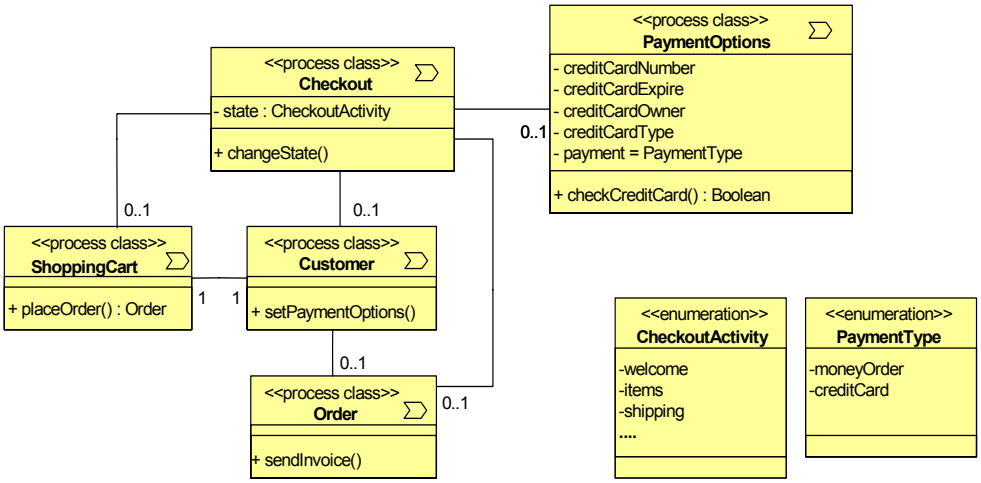

Figure 11. Process Structural View of the Checkout Process in www.amazon.com

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 (e.g. the attribute *state* of the class *Checkout*) as shown in Figure 11 or it is derived from process classes in the transitive closure concerning associations of the particular process class. Such an attribute *state* provides information for a re-initiation of the process after an interruption *(states suspended* or *cancelled)* 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.

The *process flow model* depicted in Figure 12 is a refinement of the process model at analysis level (see Figure 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 [23]. The process flow for a subactivity state is captured in another process flow model, i.e. activity diagram.

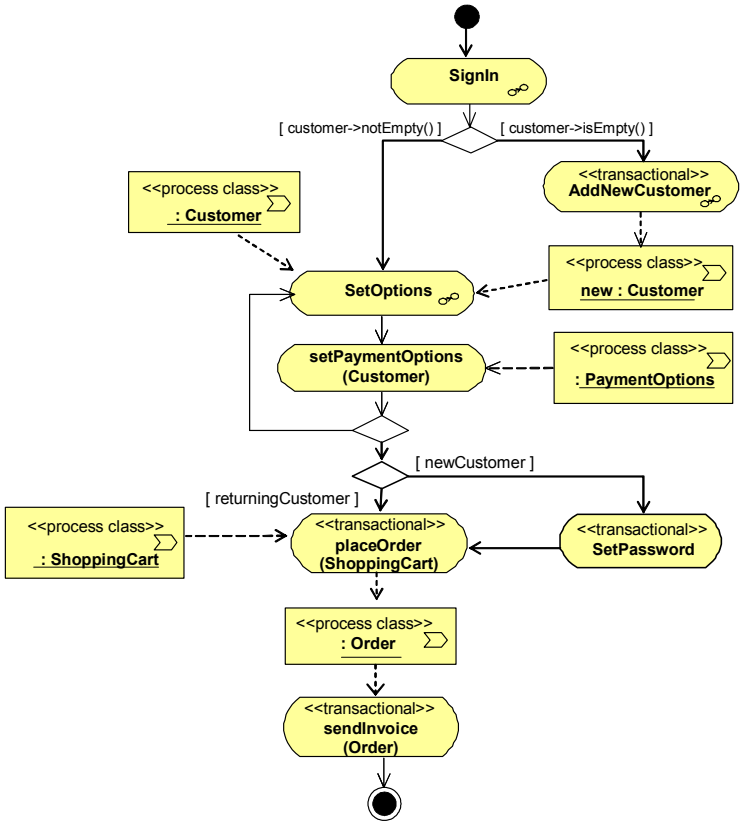

Figure 12. UWE Process Flow Model of the Checkout Process in www.amazon.com

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 *printInvoice*. 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 a call state is shown in the UML notation as the name of the operation along with the name of the classifier that hosts the operation in parentheses under it.

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 Figure 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. Figure 13 shows the presentation structure view for our example the Checkout process.

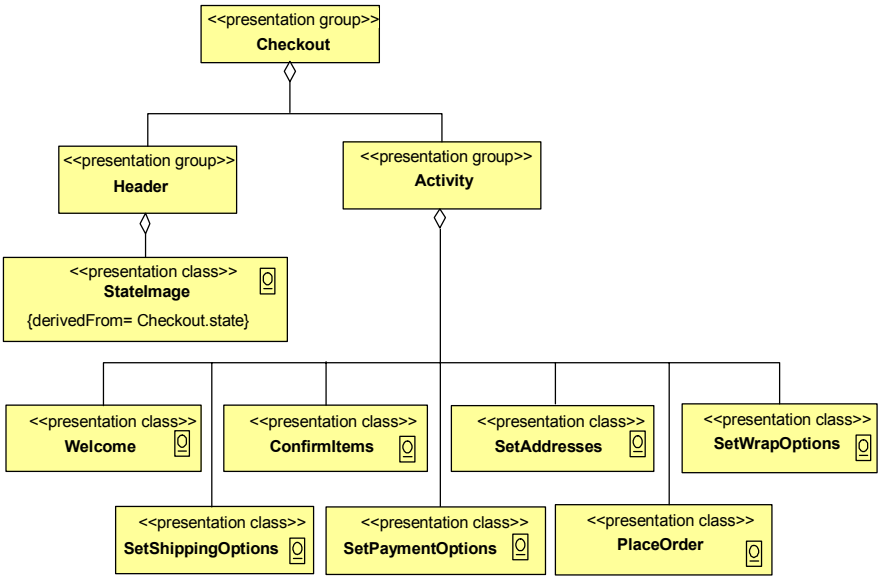

Figure 13. Presentation Structure View of the Checkout Process

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 the stereotyped classes we can observe at Figure 13, i.e. «presentation group», «presentation class» and «page». The semantic of these stereotyped classes is defined as follows:

- «presentation group» stereotyped classes are used to model the presentation sub-structure, e.g. as a set of alternatives or sub-groups. They aggregate a list of sub-locations.
- The 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.
- «page» stereotyped classes are defined as a specialization of presentation group and are used to model the smallest presentation unit that can be presented to the user.

As shown in Figure 13 the presentation group *Checkout* is divided in two sub-groups: exactly one *Header* element and one group of alternative activities – called *Activity* – whereas only one of these activities of the checkout process is simultaneously presented to the user for the interaction. 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*.

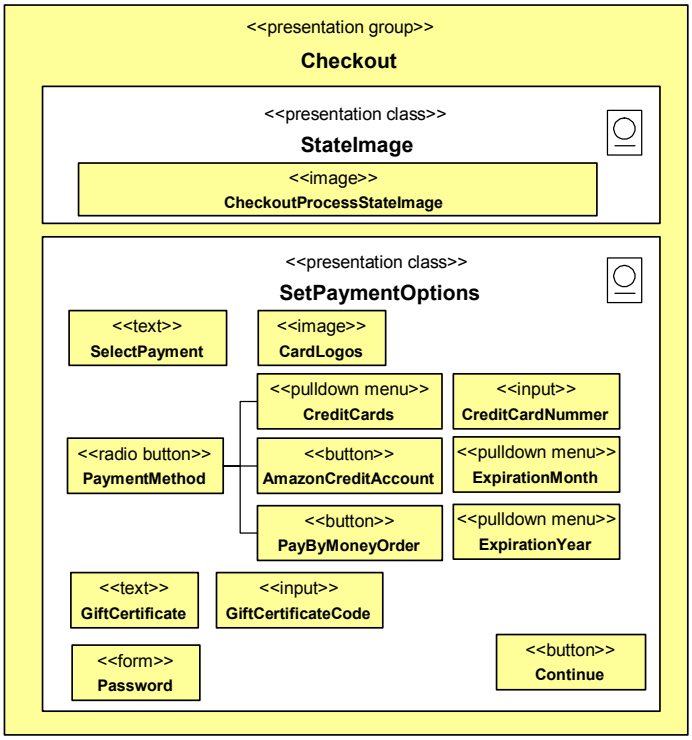

Figure 14. UI Elements View for the Presentation Class *SetPaymentOptions*

Figure 14 depicts the detailed *user interface view* of the presentation class for the *PaymentOptions* process. 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 presentation group *Checkout* together with the presentation class *StateImage* 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 *CreditCardNumber* 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.

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

6 Evaluation

Perhaps one of the most important contributions of this work is that, to our knowledge extent, it is the first time that two different proposals have worked together in the definition of a set of common concepts and modeling elements to cover Web Application needs. Previous experiences show, from our point of view, that extending proposals in an independent manner hinder the understanding among both methodologists and practitioners. The first group is usually faced with a set of similar (but not equal) concepts expressed with different notations, together with some unique constructs whose characteristics depend on the particulars of the problems for which the solution was conceived. In this sense, the use of similar vocabulary to refer to different concepts impedes the fluid communication among researchers. In addition, the different variables taken into account when developing each method make any kind of comparison very difficult. Practitioners on the other hand lack on any kind of well established criteria to choose among methodologies, not to mention changing from one to another depending on their project requirements: the effort associated with learning a new method once the practitioner has got to grips with a first one is too high.

In this sense, however, our effort has only partially achieved its goal. Our initial intention of defining a common process model has turned after two weeks of intense discussion into a common definition of an *analysis* process model. The great differences between (mainly) the navigation models of both methods have prevented us from getting at our first aim. However, we think that even this analysis model is a big step forward, as it has forced us to define a common ontology and notation on which fruitful discussions have been held. The fact that neither method had any settled concept regarding processes worked in our favor, as our aim was never to convince the other of the suitability of each one's constructs and semantics but to find a set of them that sounded reasonable for both UWE and OO-H. In this sense, we think that we have increased the probabilities of these concepts to be sharable by other methodologies and more understandable for practitioners. Our *solomonic* decision once this first goal was achieved was to include in the design models of each proposal the conclusions reached during such discussions (concepts such as process, transaction, activity or subactivity, differences between process and navigation, and so on) according to their own constructs. These shared concepts have caused that the first steps of the design phase have turned out to be also very similar in both approaches, even if they were defined in an independent way.

We believe that the Web Engineering field requires more of such experiences where different methodologies without preconceived ideas work together, perhaps on the basis of a set of common examples, in order to unify the Web modeling ontology. Also we think that the turn towards standards such as UML will help to spread Web Engineering methods and techniques among practitioners in the same way traditional software engineers have done in the past for non Web applications.

7 Related Work

The consideration of processes in best known Web modeling proposals has been quite a late add-on. In fact this shift from Web information systems where the functionality is mainly given by navigation concerns towards higher level functional features (not exclusive of Web applications) shows in our opinion a clear symptom of the degree of matureness that such proposals have already achieved.

In order to tackle this *process* concept, some Web Engineering (WE) approaches have defined the notation and semantics needed for a new model. This approach is similar to that followed by other *traditional* software (not specific to Web) engineering approaches, which regard the business processes involved in the application as a main concern and propose a separate view for its definition. WE methods have however gone one step further, and, being devoted to the abstract modeling of application interfaces, have accompanied the definition of the business processes with that of how those processes affect the user navigation activities.

In this sense, OOHDM [21] approach is based, in the same way as UWE, on the enrichment and partition of the design space into entity classes and process classes. In OOHDM each process class wraps an activity involved in the process and forming part of a hierarchy topped by the business process activity. This top business process activity is the responsible for defining the control flow of the child activities. Similarly, the navigation schema is enriched with activity nodes that represent the output and input pages relevant for each activity. Activity nodes are represented in this navigation diagram inside a process context, and this context has a shape similar to that of a UML state diagram. Links going into/outside the process context are associated with suspend/resume/cancel actions that are controlled in the corresponding conceptual process class. This process class is responsible for capturing the activity functionality.

WebML [4] on the other hand supports the idea that, being process, data and hypertext conceptual modeling the key ingredients of the model-driven development of Web-enabled workflow systems, the process model can be represented using extensions of the data and hypertexts models, i.e. these models are enriched with Workflow Management Coalition (WFMC) activities. Furthermore, they claim, as OO-H does, that a hypertext specification that embodies the interface for executing the process activities – and that therefore respects the process constraints – can be derived from a process model. The most outstanding features of WebML are (1) the consideration of workflows where more than one actor is involved (and therefore how the corresponding hyperviews may be synchronized) (2) the possibility to control certain types of activity flows (namely those that, acting on a same object, change its state with each activity) by the assignment of objects and, last but not least, the definition of a class framework that supports the whole approach.

WSDM [9] is another well known proposal that bases the definition of a new task model on the ConcurTaskTree notation [19] and refines it to specifically deal with Web application concerns. Temporal relationships are expressed by means of operators between tasks, and for each of them an object chunk, modeling the information and/or functionality required, is defined in an extended ORM (Object Role Modeling) notation. Last, a set of task navigation models, based on the task models and the object chunks, are defined to specify how the user will be able to perform the tasks in the Web site. This navigation view includes two new process constructs: components (which must be linked to the corresponding object chunk) and process logic links, that is, complex structures that include information such as their resume/interrupt effect on the process involved. Also the definition of transactions is supported in this model. WSDM considers processes being cooperatively carried out,

and in this way still a new model, a cooperative tree (also borrowed from the ConcurTaskTree approach) is proposed.

Table 2 summarizes the most relevant features of these approaches together with OO-H and UWE. In this table, P stands for *partial,* meaning that the feature is only partially supported by the method.

Features	OOHDM	WebML	WSDM	UWE	OO-H
Support for different actors	√	√	√	√	√
Support for sequence, concurrency, optional tasks	√	√	√	√	√
Support for iteration/recursivity of tasks	√	√	√	√	√
Support for task decomposition		√	√	√	√
Support for task synchronization (single or multi-user)		√	√	√	√
Support for definition of deactivation tasks			√		
Support for arbitrary information exchange between activities	√	√	√	√	√
Support for transactions	√	√	√	√	√
Support for process suspension/resuming/canceling	√		P	√	√
Support for persistence	√				
Support for multiple processes running at a time	√				
Explicit process model		√	√	√	√
Explicit separation of navigation and process concerns			√	√	
Documented class framework		√			
Compliance to standards	P	P	P	√	P

Table 2: Comparison of Process Features in Web Methodologies

In this table we observe how some features (such as support for processes that involve more than one actor or the definition of sequence, concurrency, optional tasks, iteration and recursivity of tasks) are a common feature of the different approaches. If we now regard at the differentiating ones, we can distinguish between those that are due to the notation used to define the process model (ConcurTaskTree in the case of WSDM, WFMC in WebML, UML in OO-H/UWE and a proprietary notation in OOHDM), and those due to the emphasis put in each approach over each process feature. Regarding notation, ConcurTaskTree provides WSDM with capabilities such as task decomposition, task synchronization, deactivation of tasks, information exchange or certain types of activity suspension (valid as long as the activity that causes such suspension is also present in the task tree). From them, deactivating tasks are a unique feature of this approach.

Task decomposition, synchronization and information exchange are also provided by UML to OO-H and UWE. OO-H additionally considers the definition of a state diagram associated with each process to reflect the link activations that may cause such process to be cancelled, suspended or resumed during the user navigation. The same feature is supported in OOHDM by means of its activity nodes and in UWE by the state variable of the process classes.

In all the considered approaches, transactions require additional mechanisms that have been provided by every approach, probably due to the commonality in Internet of transactional processes (e.g. purchase or bank processes) that could not be tackled until now.

Persistence of processes (in order to support undoing activities) and support for multiple processes running at a time (process orchestration) are advanced features only considered in OOHDM, while only WebML has explicitly defined a class framework that supports the defined structures.

Last, and regarding the method itself, only UWE and WSDM define new diagrams for both structure and navigation through processes. WebML and OO-H define a new conceptual process diagram, but rely on their traditional navigation constructs (with some additional restrictions) for the materialization of such process. Finally, OOHDM extends both its structure and navigational model to support the new concepts. It is also important to note how UWE is the only approach that has not added any proprietary notation to define structure nor navigation through the process.

Other approaches, less relevant from the point of view of this work, include Araneus2 [1], which was the first approach to define the mechanisms to allow the grouping of activities into phases, Wisdom [18], which proposes a UML extension that includes the use of a set of stereotyped classes, or [17], which centers on the extension of UML to better specify general (not exclusively Web-related) interaction design.

Last, we would like to stress the fact that many of the proposals studied assure that their process models could be equally useful if used in conjunction with a different methodology. This feature proves, from our point of view, the clear separation of concerns that is being achieved in most Web methodologies.

8 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, 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 and better documentation 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), as the reader may have noticed in Section 6. Such differences make very difficult to reach general agreements or widespread the use of the individual methodologies, as different research events have 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 a standard notation only in first phases of the modeling process and relies on proprietary constructs, which clear Web-specific semantics, to tackle the design models. Furthermore OO-H keeps the set of diagrams to a minimum in order to diminish the modeling effort and 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 the UWE metamodel that is defined as a conservative 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 strictly performed according to the extension mechanisms provided by the UML (stereotypes, tagged

values and OCL constraints). In addition, UWE focus on a systematic development process, although 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 [4] for defining a separate model to address analysis 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 not only analysis but also design process models, such as customer sign in and checkout, in different contexts or applications. Furthermore, it eases the maintenance and Web application evolution.

9 Future Work

Considerable work still remains to be carried out in this area, both in terms of further automation of the proposed processes, and in more detailed exploration of each model. In particular, we plan to include the extension in 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 constraints used for model checking in the tool ArgoUWE.

Moving towards a common solution for modeling business processes in the development of Web applications we have to analyze how the models we proposed in this work can be applied in the context of other Web methodologies. For comparison purposes it also would be helpful to build the metamodel of each methodology.

Acknowledgements

The authors wish to acknowledge the collaborative funding support from the European Union within the IST project AGILE – Architectures for Mobility (IST-2001-32747), the German BMBF-project GLOWA-Danube (07GWK04), and the Spain Ministry of Science and Technology, project number TIC2001-3530-C02-02.

We also wish to acknowledge the fruitful comments of the participants of the International Workshop on Web-Oriented Technology (IWWOST´03) in Oviedo (Spain) on the preliminary version of this work. In particular, we wish to thank the valuable suggestions of Daniel Schwabe.

.

References

1. Paolo Atzeni, Alessio Parente. (2001). Specification of Web Applications with ADM-2. 1st International Workshop on Object Oriented Software Technology. Valencia, Spain.
2. Luciano Baresi, Franca Garzotto, Paolo Paolini.(2001). Extending UML for Modeling Web Applications, 34th Hawaii International Conference on Systems Sciences.
3. Hubert Baumeister, Nora Koch, and Luis Mandel. (1999). Towards a UML Extension for Hypermedia Design. 2nd Conference on the Unified Modeling Language (UML'99), LNCS 1723, Springer Verlag, 614-629.
4. Marco Brambilla, Stefano Ceri, Sara Comai, Piero Fraternali. (2002). Specification and Design of Workflow-Driven Hypertext. Journal of Web Engineering, Vol. 1, No. 2, 163-182.
5. Cristina Canchero, Jaime Gómez. (2002). Advanced Conceptual Modeling of Web Applications: Embedding Operation Interfaces in Navigation Design. 21th International Conference on Conceptual Modeling. El Escorial, Madrid.
6. Cristina Cachero. (2003). OO-H: Una extensión a los métodos OO para el modelado y generación automática de interfaces hipermediales. Available at http://www.dlsi.ua.es/~ccachero/pTesis.htm
7. Stefano Ceri, Piero Fraternali, Mariestella Matera. (2002). Conceptual Modeling of Data-intensive Web Applications. IEEE Internet Computing 6 (4): 20-30.
8. Olga de Troyer, Sven Casteleyn. (2001). The Conference Review System with WSDM. 1st International Workshop on Object Oriented Software Technology. Valencia, Spain.
9. Olga De Troyer, Sven Casteleyn. (2003). Modeling Complex Processes for Web Applications using WSDM. 3rd Int. Workshop on Web-Oriented Software Technology (IWWOST´03), Oviedo July 2003, 1-12.
10. María José Escalona, Nora Koch. (2003). Ingeniería de requisitos en aplicaciones para la Web: Un estudio comparativo, Conference IDEAS'03.
11. 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.
12. Ivar Jacobson, Magnus Christersen, Patrik Jonsson, Gunner Overgaars. (1992). Object-oriented Software Engineering: A Use Case Driven Approach. Addison Wesley.
13. Robert Kahn, Charles Cannell. (1957). The Dynamics of Interviewing; Theory, technique, and Cases, New York, Wiley.

14. Alexander Knapp, Nora Koch, Flavia Moser, Gefei Zhang. (2003). ArgoUWE: A CASE Tool for Web Applications. 1st Int. Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE´03) at OOIS 2003, Geneva, Switzerland.
15. Nora Koch, Andreas Kraus. (2002). The Expressive Power of UML-based Engineering, 2nd Int. Workshop on Web-Oriented Software Technology (IWWOST´02). CYTED, 105-119, Málaga, Spain.
16. Nora Koch, Andreas Kraus. (2003). Towards a Common Metamodel for the Development of Web Applications. 3rd Int. Conference on Web Engineering, LNCS 2722, Springer Verlag.
17. Panos Markopoulos. (2000). Supporting Interaction Design with UML. TUPIS Workshop at the UML'2000.
18. Nuno Nunes, José Cunha. (2000). Towards a UML Profile for Interaction Design: The Wisdom approach. 3rd Int. Conference on the Unified Modeling Language (UML´2000), A. Evans and S. Kent (Eds.). LNCS 1939, Springer Verlag, 100-116.
19. Fabio Paternò. (2000). Model-Based Design and Evaluation of Interactive Applications. Springer Verlag.
20. 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.
21. Gustavo Rossi, Hans Schmidt, Fernando Lyardet. (2003). Engineering Business Processes in Web Applications: Modeling and Navigation Issues. 3rd Int. Workshop on Web-Oriented Software Technology (IWWOST´03), Oviedo July 2003, 81-89.
22. Daniel Schwabe, Luiselena Esmeraldo, Gustavo Rossi, Fernando Lyardet. (2001). Engineering Web Applications for Reuse. IEEE Multimedia. Special Issue on Web Engineering, 01-03, 20–31.
23. UML 1.5 Standard, OMG (2003). www.omg.org
24. Roel Wieringa, Rik Eshuis. (2002). Verification Support for Workflow Design with UML Activity Graphs. International Conference on Software Engineering (ICSE 2002), pages 166-176.