Engineering Business Processes in Web Applications: Modeling and Navigation issues

Gustavo Rossi Lifia. Conicet, Argentina
gustavo@sol.info.unlp.edu.ar

Hans Schmid. F.H. Konstanz, Konstanz, Germany
schmidha@fh-konstanz.de

Fernando Lyardet. Darmstadt University of Technology, Germany
fernando@tk.informatik.tu-darmstadt.de

ABSTRACT

In this paper we discuss several issues related to the introduction of business processes in the Web Engineering life cycle. We first argue that business processes have been so far neglected by modeling and design methodologies treating them as by-products of conceptual and navigational design artifacts, and as a consequence introducing different design and usability problems in the final products. Next, we introduce a design model for treating processes as first class citizens during Web application modeling and design. In the core of the paper we show that the introduction of processes into Web applications gives rise to a set of challenging design and navigation issues. We analyze these issues and propose modular and reusable design solutions. We finally present some concluding remarks and further work related with business process description and implementation.

Categories and Subject Descriptors
D.2.2 [Design Tools and Techniques]: Object-Oriented Design Methods

General Terms
Documentation, Design, Reliability, Human Factors

Keywords
Business Processes, Web Applications, Navigation, OOHDM

1-INTRODUCTION

Web applications have evolved from Web sites showing static information to complex software providing diverse functionality in domains such as e-commerce, distance learning, collaborative computing, etc.

In particular, a new generation of Web applications has emerged: these applications use the Web platform for supporting and executing business processes and workflows such as in rental and reservation services, remote process monitoring, virtual auctions, on-line insurance policies, etc.

However, modeling and design methods (such as WebML [2] or OOHDM [13]) have not evolved to reflect these new requirements. Though business processes have characteristics that are quite different from that of a pure hypermedia application, they are usually treated as by-products of a navigation sequence and not modeled and designed explicitly. The consequences are usability problems and erroneous results of business process execution.

New trends in Web applications like information and process personalization just complicate matters more.

In this paper we present some key ideas related with the engineering of business processes in Web applications. Our main contribution is a systematic approach that helps to cope with those problems arising from the interplay between navigation and process execution. We achieve this objective by extending the OOHDM meta-model in a way that we can solve all these problems in a modular, systematic way.

The structure of the paper is as follows: we first show why processes should be treated as first class design primitives. We next show how to incorporate processes as basic constructs in a Web design methodology. We discuss some particular problems originating from a free intermingling of business processes and navigation in particular the need to customize processes to deal with different user profiles. Though we use the OOHDM approach as a basis for our discussion, ideas underlying processes and activities can be applied equally to other design methods such as WebML [2] or W2000 [1].

2-INTRODUCING BUSINESS PROCESSES IN WEB APPLICATIONS. THE PROBLEMS

Many modern Web applications involve a set of activities that users have to follow carefully to carry out a task; examples are the checkout process in electronic stores such as www.amazon.com, the registration process such as in www.sun.com or the purchase process in www.expedia.com.

In a business process like the checkout process, the user has to go through a predefined sequence of activities: login, confirm the selection made, enter the address or other relevant personnel data, select some delivery options, ways of payment, etc., before the process is completed successfully.

This kind of processes is often emulated as a sequence of navigation steps, both with regard to modeling and implementation. That means each activity is mapped to one (or more) Web pages that are treated in the same way as other navigational objects (products, plane itineraries, etc.). Though it is possible to simulate, in a certain way, a business process like the checkout process using navigation, we may experience some usability or more serious problems due to the different semantics of hypermedia navigation and business process execution.

To make this discussion concrete, consider the following scenario: the user leaves the checkout process by following some link, for example to explore the products in the shopping cart, or to read about the purchase conditions (in Amazon, this scenario may arise...
while confirming the items to be bought). This may raise the following problems:

- **User disorientation**: "What happens if I add the product to the cart again? Should I start the checkout again?"

- **Inconsistent process states**: in which state does the previous process remain if the user does not return immediately?; for example, what happens if he begins shopping again and wants to checkout once more?

- **Inconsistent or unpredictable share objects states**: suppose that while exploring the product, he adds it to the cart; will this product be considered in this process? If he backtracks to the confirm items page, should this new product appear? Is the current list accurate?

Treating business processes just as navigation sequences also prevents the user from dealing with more than a process at a time. Suppose he is booking a flight, and before finishing he wants to rent a car from the destination airport: an application like www.expedia.com urges him to navigate to the second process, which is just an ordinary page, and then backtrack to the point in which the booking was left instead of just resuming the suspended process (e.g. by selecting the “flights” icon).

Last but not least, as the logic of activities and the order of execution are tightly coupled with the navigation objects, designers are not able to reuse the same activity in different processes easily.

While one may argue that these are implementation problems, e.g. we can solve some of them by preventing the user from leaving the checkout pages or by performing some run-time checking, they are caused by the fact that the semantics of navigation are completely different from the semantics of business processes. Business processes cannot be modeled and designed adequately by hypermedia primitives and navigation semantics. To solve the described problems, we have to treat business processes as first-class citizens in Web applications.

A business process [10]:

- Drives the user through its activities. That means that it defines the set of activities to be executed, and the control flow among them. In contrast, navigation is user-driven and random.

- Has a state that consists of the current activity, including the knowledge if it is active or suspended, and the previously performed activities (with a simple control flow, these can be implied from the current activity). As a consequence, the process state does imply which activities you have already executed (i.e. the history), and also which activities you have to perform subsequently. In contrast, the state of navigation is usually represented by the Web page displayed by the browser.

Thus, for achieving a correct execution:

- A business process has to keep internally its complete state; the current node on the screen does not contain sufficient state information.

- When a user navigates out of a business process like checkout or hotel booking, it must be clarified if the user wants to terminate the process or to suspend it and resume it at a later time. If the user wants to suspend it, the state of the process must be stored and conserved. In this way, if the process is resumed, it can be resumed in the state it was, when it was suspended. Since the business process defines all activities through which to go, it will guide the user after resuming the process through the pending activities.

We claim that all these differences must be taken into account, starting from the early modeling stages and made explicit in the design documents. Regarding design, we should be able to describe the set of activities and the control flow that form a business process, if it can be suspended and resumed and how navigation takes place from and to a process, in order to provide the implementer with a precise blueprint of the application behavior.

### Table 1: The hypermedia vs. the business process paradigm

<table>
<thead>
<tr>
<th>Activity</th>
<th>Hypermedia</th>
<th>Business Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure</strong></td>
<td>A flat network of nodes connected by links</td>
<td>Process composed of activities that are basic or composed</td>
</tr>
<tr>
<td><strong>Control</strong></td>
<td>The user always has control on the sequence of visited pages</td>
<td>The process defines which is the next activity to be performed. Activities are often performed sequentially, though the control flow may be more complex</td>
</tr>
<tr>
<td><strong>Leaving page/Activity</strong></td>
<td>A page is left by just selecting an anchor and navigating a link. Its state does not change. There is no notion of completion.</td>
<td>When an activity is left it must be clear if it is completed, suspended or aborted. The notion of activity completion is essential in the business process paradigm</td>
</tr>
<tr>
<td><strong>Resuming/Undoing</strong></td>
<td>Backtracking to a page (for example by clicking the back button) just means navigating to the page again</td>
<td>Returning to an activity resumes it at the current suspended state; Undoing an activity must be explicitly indicated</td>
</tr>
</tbody>
</table>

As a summary, in Table 1 we describe the differences between the hypermedia paradigm (on which the Web has been based until today), and the business process paradigm (that plays an important role in newer Web applications).

### 3-RELATED WORK

Web design methods like OOHDM [12], WebML [2,3], and W2000 [1] model and design a Web application using a clear separation of concerns. Usually, they involve a presentation model, a navigational model and an application domain model.

The navigational structure is described using primitives such as nodes, links, access structures, etc. W2000, and WebML describe the entities of the application domain with primitives based on the entity-relationship approach, while OOHDM describes them using an object-oriented class model.

Features of modern Web applications that pass beyond
hypermmedia-based navigation are modeled by operations. W2000 defines operations that may change the state of application objects, and describes the state of navigation objects with respect to operations. It considers the operational interaction paradigm as coexisting with the navigational interaction paradigm.

OOHDM, meanwhile, provides operations as methods of application domain objects that are called from methods defined in nodes.

WebML considers operations as interactions with “external” applications and models the interaction just by indicating input/output parameters. Operations are useful to describe and model an action that is related to and triggered from a single node, like adding a product to a shopping cart. However, they are not enough to express the semantics of business processes that consist of a sequence of activities, like the checkout process. We next present our approach for modeling and designing processes in Web applications.

### 4-MODELING AND DESIGNING BUSINESS PROCESSES IN WEB APPLICATIONS

In this section, we present an approach for modeling and designing processes in Web applications. We have extended the OOHDM meta-model by adding processes and activities both to the conceptual and navigational model. While processes at the conceptual model deal with the flow of activities, activity nodes provide the modeling mechanisms for managing the interplays between navigation and process execution, such as process contexts, process suspension and resuming, etc.

**OOHDM** focuses on three different design concerns: conceptual or application modeling, navigation design and interface design. During conceptual modeling, domain objects and application functionality are described using UML primitives. The navigational model describes the objects that the user will perceive with nodes and the navigation topology using links and indexes; node classes represent a view of conceptual classes, while links represent the hypermedia counterpart of application relationships.

**OOHDM** considers that navigation always occurs in a context; for example you explore a CD in the context of the checkout process or you may explore it while searching for CDs of a particular group; it may be desirable that particular nodes (e.g. CDs.) exhibit different information or anchors according to the context in which they are accessed. This variability is expressed in OOHDM by specifying Navigational Contexts.

Navigational Contexts have been extensively used to manipulate sets (books of an author, CDs in a musical genre, etc) and to provide intra-set navigation facilities, allowing the user to move to the next or previous object in the set. In order to define context-specific information, OOHDM uses InContext classes that act as Decorators [5] of the corresponding navigational object; InContext objects add that information dinamically to the decorated node when it is accessed in the corresponding context. Later in this paper, we show how we used this feature to solve the problems occurring when the user leaves a process to explore some node.

The OOHDM viewing mechanism allows describing different node classes and linking topologies for different user profiles; this kind of customization can be realized during design time (static) or even on an individual basis during the application’s execution (dynamically).

Describing the full OOHDM syntax is outside the scope of this paper, for more information see [12]. OOHDM’s customization features are described in [13].

### 4.1 INTRODUCING PROCESSES IN THE CONCEPTUAL AND NAVIGATIONAL MODELS

We introduce processes and activities in the OOHDM conceptual schema and activity nodes in the OOHDM navigational schema. For this purpose, we partition the conceptual design space in entities and processes, and the navigational design space in navigational and activity nodes. A discussion on heuristics for achieving this partition is outside the scope of this paper and is addressed in [11].

Though activities are not supposed to be always interactive, i.e. they may not need a user interface, we focus on those that need some kind of user involvement. For those activities we need to build activity nodes, in the same way as we do with all objects that are going to be perceived and navigated by the user; therefore, activity nodes are described as views on activity objects. As shown later in this paper, modeling activities as first class citizens in both schema not only help us to solve some of the previously mentioned problems but also to introduce process customization, multiple process execution, etc.

Figure 2 and 3 show the OOHDM conceptual and navigational schema of a simple CD store. At the bottom of Figure 2, you see entities like CD, or ShoppingCart, and at the left of Figure 3, entity nodes like CD Node or ShoppingCart Node. At the top of Figure 2, you see process and activity classes, and at the right of Figure 3, process and activity node classes.

The conceptual model shows that a business process like CheckOut (Figure 2, top) is typically composed of several activities like the Login Act, ConfirmItems Act, ShippingAddress Act, DeliveryOptions Act, ConfirmAll Act, and PaymentOptionsAct. This is represented by an aggregation relationship in the conceptual schema.

An activity is either basic, like the LoginAct, ConfirmItemsAct, etc., or (recursively) composed by other activities, like CheckOut and PaymentOptionsAct, following the composite pattern [5]. A composed activity without a parent activity like CheckOut represents a business process.

In the navigational model, activity nodes, like LoginAct Node and ShippingAddressAct Node (Figure 3 right) play the same role as navigational nodes. They describe, in an abstract way, the visible attributes, anchors and operations with which the user will interact during process execution. The interface of an activity node (for example a Web page) will contain buttons like “ok”, “commit”, “cancel”, or “next” that control the input processing of an activity and the control flow to a subsequent activity.

Activity nodes like LoginAct Node are shown in the context of the corresponding process node to which they belong (a composite in OOHDM), like the CheckOutAct Node. This is indicated by drawing the activity nodes within the box of the process that forms the context in which they are executed.
4.2 SPECIFYING CONTROL FLOW WITHIN A PROCESS

A business process defines the sequence in which its child activities should be executed. For example, CheckOut defines that the activities Login Act, ConfirmItems Act, ShippingAddress Act, DeliveryOptions Act, etc. are executed in a strict one-way sequence. We describe the control flow on the conceptual level by adding an UML activity diagram to the conceptual schema. The logic of the business process may involve checking the customer profile or taking decision based on user’s selections. However, in simpler cases (as in the example above) we can indicate the control flow among the activity nodes of a process, in a similar way as in a UML state diagram (see the CheckOutAct Node in Figure 2), with special initial and final diagram nodes. Directed edges among activity nodes show the possible flow of control among the activities. This notation is particularly useful when different navigational views may implement different sequence of activities; however when this kind of customization is achieved dynamically (as discussed in Section 5.3), or when the control flow follows a more complex pattern it is not possible to indicate the process flow in the diagram. Notice that there is a subtle difference between navigation links followed randomly by the user and the state diagram arcs that indicate the process logic. In our example, the edges show a one-way sequential control flow from the LoginAct Node over ConfirmItemsAct Node etc. to PaymentOptionsAct Node. The latter contains a nested flow leading from the sub-process initial node to its final node either via CreditCardAct Node or via MoneyOrderAct Node. The interface of each of these nodes may contain a “next” button that lets the user move from an activity like ShippingAddress Act to the subsequent activity like DeliveryOptionsAct, except for the ConfirmAll Act Node, which will contain a “done” button. Following OOHDM behavioral semantics, the “next” button will trigger a node’s method, which in turn will send a message to the corresponding conceptual activity; the activity will itself communicate with the CheckOut object; this object will then dispatch the next activity.

Alternatively, CheckOut might define a control flow called a two-way sequential control flow which is well known from wizards or assistants, used e.g. for installations processes. It allows a user also to go back from an activity like ShippingAddress Act to the preceding activity like ConfirmItemsAct by pressing a “previous” button that triggers a back action. Thus, a user may correct input data entered in a preceding activity without having to redo the whole process. Process and corresponding activities can be implemented in a straightforward way using a state-machine approach, e.g. using the State design pattern [5]. According to its current state i.e. which activity is being executed, whether the process is active or suspended, the process reacts to messages such as “next”, “done”, etc. The more complex case in which process execution depends upon the user profile is discussed in Section 5.3.

4.3 STARTING, PROCESSING AND COMPLETING A PROCESS/ACTIVITY

The operations supported by an activity are shown in the conceptual schema (left off in figure 2 for space reasons). Each activity must support an operation to start processing, and we recommend also supporting a cancel operation so that a user can cancel a business process at any time. Suspend and resume operations may be supported if required.

A business process may be started by an user action triggering an operation on a navigation node. For example the “checkout” method in the shopping cart which upon activation sends the message “start” to the corresponding Checkout process object. The Checkout parent activity starts then the Login Act child activity which will return control to the Checkout Act after the user has successfully completed or cancelled it (e.g. by pressing a button of the corresponding Activity Node interface). After a successful login, the CheckOut Act starts and passes control to the ConfirmItems Act child activity, until this returns control, and so on. Composed activities have the responsibility to pass the control among its child activities. A child activity does not pass control to another sibling activity when it terminates but returns control to its parent activity.

Decoupling the control flow among activities from the activity objects, allows activity classes to be reused in different processes. For example the PaymentOptions Act activity class might be used in a different application such as plane tickets selling, though the parent checkout process may differ completely from the parent plane ticket ordering process.
4.4 SUSPENDING AND RESUMING A PROCESS

A designer may allow the user to suspend a business process, leave it for navigation, and to resume it later. Consider the example of the checkout process: you may want to allow the customer to suspend the ConfirmItems activity for navigating to the CD node and look at details of a CD to see if he wants to confirm it. In this case navigation should suspend the process explicitly to avoid previously mentioned inconsistencies.

We have modified the OOHDM meta-model in order to support the business process paradigm. A process may be suspended; in the case of the checkout process, we can navigate to a CD page from the ConfirmItems activity. Should the user be allowed to add the CD to the shopping cart again, while the checkout process is suspended, and if so what would be the semantics of this action? We may not want to allow this operation since the checkout process may already have created the order with the items currently in the shopping cart. Therefore, an item added during navigation would not be taken into account when the checkout process is resumed, thus confusing the customer. More generally, when a business process is suspended, a user should not perform operations that modify the state of resources being used by the process.

A solution to this problem is to restrict navigation from activity nodes: in our example the ConfirmItems activity should not provide any link, e.g. the user can see the names of the CDs but cannot navigate to them. This solution may not always be appropriate, since it prevents users from exploring other pages (which is the most appealing feature of the Web). A better solution is to remove the action: “add to cart”, from a CD node when we access the CD after leaving the checkout process. We can achieve this objective easily by combining processes with navigational contexts.

Every process defines a navigational context: this means that when a user suspends a process, navigation occurs in the navigational context of this process. The navigational context of a process specifies, in the same way as a usual navigational context, which restrictions or additions apply to a node when it is accessed in the context of this process. We do this by defining InContext classes that “decorate” the corresponding base class with the new features (restrictions or additions).

In this way, we can make a “fine-tuning” of the features of nodes when accessed in the context of a business process or even from a particular activity in the process. In Figure 5 we present the context diagram for CDs (using the OOHDM notation), by incorporating the checkout context. The diagram specifies that a CD can be accessed while exploring CDs of a group or CDs of a particular genre, while accessing the shopping cart and while performing the checkout process.

In Figure 5 we show the corresponding InContext class for a CD in the context of the checkout process. Similar to a CD in a shopping cart context, it modifies the “add to cart” method; further, we have added a method for returning to the checkout process.

5.1-NAVIGATING IN A PROCESS’ CONTEXT

In Section 2 we have shown the problem that arises when leaving a process using a navigation link. For example in the checkout process, we can navigate to a CD page from the ConfirmItems activity. Should the user be allowed to add the CD to the shopping cart again, while the checkout process is suspended, and if so what would be the semantics of this action?

We may not want to allow this operation since the checkout process may already have created the order with the items currently in the shopping cart. Therefore, an item added during navigation would not be taken into account when the checkout process is resumed, thus confusing the customer. More generally, when a business process is suspended, a user should not perform operations that modify the state of resources being used by the process.

A solution to this problem is to restrict navigation from activity nodes: in our example the ConfirmItems activity should not provide any link, e.g. the user can see the names of the CDs but cannot navigate to them. This solution may not always be appropriate, since it prevents users from exploring other pages (which is the most appealing feature of the Web). A better solution is to remove the action: “add to cart”, from a CD node when we access the CD after leaving the checkout process. We can achieve this objective easily by combining processes with navigational contexts.

Every process defines a navigational context: this means that when a user suspends a process, navigation occurs in the navigational context of this process. The navigational context of a process specifies, in the same way as a usual navigational context, which restrictions or additions apply to a node when it is accessed in the context of this process. We do this by defining InContext classes that “decorate” the corresponding base class with the new features (restrictions or additions).

In this way, we can make a “fine-tuning” of the features of nodes when accessed in the context of a business process or even from a particular activity in the process. In Figure 5 we present the context diagram for CDs (using the OOHDM notation), by incorporating the checkout context. The diagram specifies that a CD can be accessed while exploring CDs of a group or CDs of a particular genre, while accessing the shopping cart and while performing the checkout process.

In Figure 5 we show the corresponding InContext class for a CD in the context of the checkout process. Similar to a CD in a shopping cart context, it modifies the “add to cart” method; further, we have added a method for returning to the checkout process.

5-FURTHER DESIGN ISSUES

Although there are many Internet applications that may be completely or at least mainly driven by business processes, we are interested in those applications that combine both the navigation/hypermedia paradigm and the business process paradigm.

The introduction of activity objects and nodes solves the problems related with the specification of control flow within activities and among activities and navigation objects; however there are still some problems derived from the combination between process execution and navigation. In the following sub-sections we address the most important issues related with this combination.

Figure 4 Contexts in which we can access a CD
5.2 LANDMARKING PROCESSES

As mentioned in Section 2, there may be situations in which the user may like to trigger more than one business process at a time; for example while booking a flight, he wants to rent a car or be sure that he can have a room in a hotel. Pure navigation simply doesn’t work well: the user has to leave the process, begin another process and then return to the initial process by backtracking. In the meantime he can suffer disorientation, or experience other problems discussed in Section 2.

An interesting solution consists in using the spirit of the Landmark pattern whose intent is to “provide easy access to different though unrelated subsystems in a Web application” [9]. Landmarks provide direct access to outstanding nodes in complex applications; different sub-stores in sites like www.amazon.com are accessed using landmarks, thus simplifying access to the store from every single node in the site.

If we model flight booking, hotel reservation or car rental as processes we can apply the principles that we discussed in Section 4.4: when we leave a process (e.g. flight booking), the process remains suspended; we can then start another one (e.g. car rental) and return to the previous one, expecting a “safe” state, i.e. that we can finish with the process without inconsistencies. However, even using activity nodes with the semantics discussed in Section 4, we still need an easy way to “switch” control among processes. This is provided by defining Process Landmarks, i.e. outstanding and easy to access process nodes with corresponding anchors and links that are visible from every other node in the application. Notice that many of these customized behaviors may require modifying the application code, e.g. the process objects; thus, the way in which we design processes and activities is critical to achieve modular and painless software evolution.

Extending this functionality to support process landmarks is straightforward; first, we have to design each piece of functionality (flights, car rental, cruises, etc) as a process with corresponding activities; next we need to follow the semantics for suspending and resuming a process that we discussed in Section 4.4. We use the OOHDM notation for Landmarks as shown in Figure 7. Flights and Hotels are process landmarks as indicated with the “L” in the top right of the corresponding class. For the sake of conciseness, we do not show the corresponding conceptual schema. Notice that given the semantics of nodes in OOHDM (in particular activity nodes) we can manage multiple processes with landmarks without modifying the conceptual schema, i.e. even if we enrich the navigation topology with process landmarks, we do not need to care about the base application model.

Figure 5 InContext Class for CD in checkout process

Figure 6: Landmarks in Expedia.com

Figure 7: Modeling Process Landmarks

Customization has become hype in areas such as electronic commerce, and we can find hundreds of applications that claim to be fully customizable to different user profiles or individuals. It is possible to find many different personalization patterns: for example we may personalize the links allowing different users to explore different pages such as Amazon recommendations; we can adapt the contents of a page to let different users access individualized contents, etc. Regarding processes, there are many different ways of personalizing them; for example in www.amazon.com a new customer has to follow the previously mentioned step by step procedure (one page for each activity), while a registered customer just confirms all data in one page as shown in Figure 8; a customer can sign for what is called “one-click” check-out in which the process is “automatic”; etc.

A more elaborated customization policy may provide special offers for holders of a particular credit card; in this case we may need to “break” the normal process control flow to add a new activity to let the user select from a number of offers. Notice that many of these customized behaviors may require modifying the application code, e.g. the process objects; thus, the way in which we design processes and activities is critical to achieve modular and painless software evolution.
We have claimed elsewhere [13] that personalization should be addressed using a design more than an implementation view. This means that once we understand what we want to personalize, we have to express this personalization feature using the corresponding design primitives, before deciding how it will be implemented.

Treating processes and activities as first class objects, and modeling them in the context of the OOHDM framework allow us to apply most of the design rules defined in [13] for seamless process customization.

The simpler example of process customization consists in providing different navigation/interface functionality for the same process or activity to customize it to different user roles. For example when an Amazon employee performs the checkout process he may be provided with different navigational options that a regular user can not see; in OOHDM this is achieved by defining different navigational schema, providing different linking topologies, one for each user role as shown in the simplified diagram of Figure 9.

Both navigational views in Figure 9 share the same conceptual model (e.g. the one in Figure 2). However, while performing checkout the customer can navigate to the CDs in the cart while the employee can also access information about benefits related with the products he is buying. Notice that this “pure” navigational customization is transparent for process and activities.

We may want to provide a simplified checkout interface for registered users like the one in Figure 8. Again, the OOHDM viewing mechanism allows specifying the ExpressCheckOut activity node containing all the information provided by the former simpler activities as shown in Figure 10. This activity provides an interface in the spirit of OOHDM composite nodes. Each component node, e.g. item List or shipping address, replaces the corresponding activity in Figure 3. Notice that each of these attributes belongs to a type (e.g. Item Node), which is itself a full-fledge node class. There is no need to define an “internal” control flow; however, if this activity is left following an anchor, the semantics of suspension and resumption remain as discussed before, as well as the notion of navigational process context.

Figure 11: Customizing different views of the same process

In Figure 11 the start method in CheckOut asks the profile to return the initial state and then starts the corresponding activity, also provided by the profile object. Notice that class Profile also contains methods for defining states and activities; in this way we avoid having to sub-class Profile for different user profiles. Finally, we might want to completely customize the control flow between activities according to the user profile. For example certain customers that paid a special fee when registering have a express delivery option, and thus they do not have to choose one of the options; or a set of cheaper products may be offered to customers paying with a Visa card and thus, a new activity has to be introduced. If we want to achieve complete process customization according to the user profile, the best solution is to decouple the control flow from the process object and delegate it to the corresponding profile object as shown in Figure 12.

In the micro-arquitectures of Figure 12, the behavior for deciding which is the “next” activity is delegated to the profile object as well as the information on the current state.
 objects) in order to separate the different concerns involved in the policies from the application code (in particular from the process appliances, etc., is described in [7]. It further decouples those policies such as those related with network connection, interface

A generic architecture for achieving more complex customization policies such as those related with network connection, interface appliances, etc., is described in [7]. It further decouples those policies from the application code (in particular from the process objects) in order to separate the different concerns involved in the problem: the customization rules, the process (and domain) objects and the profile. Our approach can be easily used in the context of this architecture just by further separating the rules that guide processes to a separate component.

6-IMPLEMENTING PROCESSES

Though outside of the scope of this paper we next detail some interesting issues related with implementing business processes in the Web environment.

A well-known alternative for implementing complex Web applications is the Model-View-Controller paradigm (MVC) [8]. It divides functionality and separates design concerns in three blocks: data persistence and behavior in the Model, presentation in the View, and interaction in the Controller. There are a number of MVC implementations available for Web applications such as Struts [6] and the host of benefits the MVC provides has been widely discussed in [8].

However, when introducing business processes, current MVC implementations fall short in coping with this new requirement. One example is the Struts implementation which although its outstanding performance in small to medium Web applications, makes it difficult to handle complex business logic; logic has to be spread on the different Action objects which tend to grow and become more complex as the application evolves. Therefore, developers tend to create separate Java beans to keep the Action objects small. As a consequence of the Struts design, the framework does not provide much assistance in organizing business transactions that require more than one interaction with the user as those shown in this paper (because they span more than one JSP page and/or Action).

This weakness unveils the fact we discussed in this paper: business processes are usually misplaced due to the lack of a clear and standard architecture and design approach, thus obscuring implementation and preventing extensive reuse.

Founded on the same conceptual ideas of design concern separation, we are working on an extension of the MVC architecture in order to prevent the coupling of business logic with either the Model or the Controller by including a fourth design tier: the Business Processes tier. In this way, we propose to separate the implementation of processes control flow from the controller objects and from the entity objects that are implemented as part of the model.

To test the proposed extension, we committed to a standard, widely available MVC implementation like Jakarta Struts. In Figure 13.a we show the standard Struts schema and in Figure 13.b our extended MVC architecture, including business processes.

![Figure 12: Decoupling process control flow for achieving customization](image1)

A generic architecture for achieving more complex customization

![Figure 13.a: Struts architecture](image2)

![Figure 13.b: Adding business processes to Struts](image3)
Solving the usability problems caused by the interplay between navigation and processes without restricting free navigation.

Overcoming state inconsistency problems resulting from the shared use of an entity by a process and by navigational nodes during suspension of the process.

Reducing the semantic gap between business processes and their design models

Easy specification of (static and dynamic) customizable processes.

Straightforward implementation using an extension of the Model-View-Controller framework.

Enabling the reuse of activities in different processes: activities such as customer login or payment options can be reused in the context of different processes thus improving the quality of design and making implementation more cost-effective.

We have used the proposed design method successfully for a number of Web applications, both in student projects and in cooperation with software houses in real world projects. Some of these applications are a customer relation management system for small and medium sized shops and companies, which embodies different business processes, a cooperative travel agency where users can share traveling opportunities, and several Web shops.

As mentioned in Section 6 we are working on the formal definition of an extension of the MVC metaphor to introduce business processes seamlessly in the implementation, in particular using well-known frameworks such as Struts.

Finally, we are also studying how to enrich our notation by using UML stereotypes both in the design process and to describe different implementation strategies as described in [4]

8- REFERENCES


