Finalizing Dialog Models at Runtime

Stefan Betermieux and Birgit Bomsdorf

Fernuniversität in Hagen, 58095 Hagen, Germany
stefan.betermieux@fernuni-hagen.de,
birgit.bomsdorf@fernuni-hagen.de
http://www.fernuni-hagen.de

Abstract. This paper proposes a dialog model for web applications aiming at flexible interface generation. The basic idea is to enable the runtime system to "finalize" the dialog structure. The overall approach follows a task-oriented, user-centered development process, where models of the users' tasks and the user-system dialog play an essential role. In our approach, these models are transferred to the run time system that allows the user to interact with the web application according to the specifications. It is based on an architecture that separates a task controller and a dialog controller, which are responsible for model execution and dialog creation. Throughout the paper, we take care of the special characteristics of web applications and show enhancements of the conceptual models and of the runtime architecture.

1 Introduction

Web sites have been developed towards highly interactive web applications. The web site visitor is no longer a mere recipient of information but a user interacting with an application, e.g., filling in data, triggering system functions, and receiving feedback from the application. Thus, web pages evolve more and more into user interfaces; besides providing content they have to support user-system interaction, also referred to as the dialog. Furthermore, up-to-date web applications, such as e-shops, include business processes. The system has to guide users through a predefined sequence of web pages through which they perform the activities of the process.

In the field of model-based design of user interfaces (e.g. [7], [20], [21], [24], [25]) well-known notations and techniques exist for developing systematically the dialog of an interactive system. They were developed, however, for modelling traditional user interfaces. Within the web modelling community ([22], [15], [13], [17], [5]), on the other hand, the interactive behavior is specified basically implicitly within the navigation model. In some approaches, similarly to the field of HCI, models describing the system from the view of the user are introduced. In WSDM [13], for example, task models are used as a high-level dialog description to guide the design of the navigation. In OOHDM [26], task descriptions are analyzed to identify the data items, which are to be exchanged between the user and the web application. In general, web modelling approaches are complemented increasingly by the investigation and specification of user goals, tasks and activities, respectively (further examples are given by WebML [8], UWE and OO-H ([10],[17])). All in all, these aspects are used as informal input to conceptual domain, navigation and presentation design. Since interaction design and feedback specification becomes more and more important, this information should not be an add-on to an otherwise data-oriented methodology. Both a data-centric view and a task- and interaction-centric view are required. However, if a modelling approach supports mainly the data and object based specification treating dialog specification implicitly only, achievement of a usable interaction design can be very cumbersome. For example, spreading pieces of interaction specification over the navigation model makes it difficult to detect common structures and patterns, respectively. From our point of view, developers have to be supported by modelling concepts that put the dialog explicitly and coherently into play.

The work presented here proposes a flexible dialog model. The whole approach follows a task-oriented, user-centered development process. However, in this paper we focus on the conceptual task model and its associated abstract user interface model. In most existing approaches these models are taken as input to subsequent development steps, within which the sites and pages are modelled in more detail aiming at the final web pages and navigation. In contrast to this, in our approach the task and abstract user interface model are passed over to the run time system. Its control component comprises a task controller and a dialog controller responsible for "finalizing" the dialog model. The objective is, to use this technique in the adaptation of web pages to various screen sizes. Implementation of the overall framework is work in progress. The scope of this paper is to introduce the basic concepts of the *flexible* dialog.

2 Task Modelling for Web Applications

Modelling of a web application usually starts with requirement specification, similarly to traditional Software Engineering. The objective is to get a picture, as clear as possible, of information and functional as well as of usability requirements. Use case diagrams are commonly in use for a first description of them, later on refined by means of, e.g., activity diagrams or task models. Like WSDM [14], we apply task modelling for this step. While in WSDM a modified version of CTT [20] is used, we adapted the notion of VTMB [4] for the concerns of web modelling. Initially VTMB was developed to support task modelling in the context of traditional user interfaces. Focussing on interactive web applications, additional concepts are added by our current work. The concepts as relevant within this paper are introduced below, whereby we concentrate on task models as applied in conceptual design.

2.1 Task Model

Throughout this paper, a task is denoted by means of a symbol as depicted in figure 1. Defining temporal relations (top right in the task symbol) and cardinality

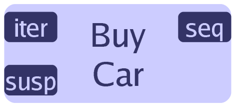

Fig. 1. Example of a task symbol (representing the task Buy Car)

(top left) is well-established in task modelling. In addition, we enriched it by the task lifespan (denoted bottom left).

Task Cardinality. Information can be attached to a task symbol to express the possible number of executing that task. Internally, minimum and maximum values are used to control at run time the cardinality constraints. To support readability of task models, cardinality information is attached to task symbols by means of expressive names.

Optional (opt). The label *opt* denotes that the task can be omitted.
$minimum = 0$

Iteration (iter). A task labeled with *iter* can be performed as often as needed.
$maximum = \infty$

Mandatory. If no label is assigned to a task symbol, the task is to be executed exactly one time, which is the default value for task performance.
$minimum = 1$, $maximum = 1$

Temporal Relations. Task relations are a basic concept regarding hierarchies of tasks. In contrast to CTTs [20], we assign a temporal operator to the relation between a parent task and its children tasks and not individually between the siblings.

The universal relation is always an aggregation of the child tasks to a parent task (is-part-of relation); a parent task is fulfilled, as soon as the sub-tasks are fulfilled according to the temporal relation given. This is a very basic definition of the possible semantics of the relation, they are refined to be more useful in the context of task models.

Sequence (seq). The subtasks of a parent task have to be completed in sequence. The second subtask can only be performed, if the first subtask is finished. This process is continued for all subtasks. The parent task is fulfilled, as soon as the last subtask is finished.

Arbitrary Sequence (arb). The subtasks can be completed in an arbitrary sequence. Only one subtask can be performed at one time. The parent task is fulfilled, when all subtasks are finished.

Selection (sel). Only one of the subtasks can be selected to be performed. The parent task is fulfilled, when that subtask is finished.

Parallel (par). All of the subtasks can be performed in parallel. There can be more then one subtask performed at the same time. The parent task is fulfilled, if all subtasks are finished.

Lifespan. In the context of web applications, task models have to cope with user originated and system originated task suspensions. This holds true for traditional applications as well, but has a slightly different meaning in the context of the web.

User originated task suspension occurs, when the user navigates explicitly to a page which would not be accessible by the current task model, for example by using bookmarks or the browsers navigation buttons. These out-of-context navigations cannot be solved automatically, but have to be addressed by the task model designer.

Due to the stateless hypertext transfer protocol (HTTP), the server needs to associate user requests to a memory area where the state of the application of a user is saved. Since the memory on the server is finite and users may possibly never return, the server needs to clean up this memory area (the user session), normally by using timeouts after the last request. Hence, *system originated task suspension* occurs, when the web application server terminates a user session which contains a running task model instance.

Either way, the task controller needs to detect those exceptional events and react to them with a predictable behavior. Instead of dictating a single task model suspension strategy, it is possible to attach those strategies to tasks, where they only affect the task and all of their subtasks. It is possible to mix strategies in a task model depending on application requirements, as long as the tasks with different strategies are on separate subtrees. The example in section 2.2 will explain this behavior in more detail.

Volatile. The default strategy for tasks without a lifespan attribute. If a suspension occurs, the states of the task and all its subtasks are discarded.

Suspendable (susp). If a task is marked as suspendable and suspension occurs, the task controller will put it and all its descendants into a suspended state. The user is offered an interface option to resume the suspended tasks.

Persistent (pers). A task marked with pers is suspendable as defined by susp and additionally will be persisted to a long term storage space (i.e. database) when a session timeout occurs. This is only possible for users which can be re-identified in the next session, for example by a name/password registration, or a site cookie. During the next session, the user is offered an interface option to resume the previously suspended tasks.

2.2 Example

Figure 2 shows a simplified model of a web application, which uses most of the above mentioned concepts. The example is taken from an on-line picture gallery where users can browse existing pictures and upload new ones. Starting from the root task *On-line Picture Gallery*, three subtasks can be invoked independently (the relation to the subtasks from the root tasks is marked as parallel). The sub task *Help* provides context sensitive help and can be re-invoked if finished (task is marked as iteration). The sub task *Browse Pictures* is subdivided into two subtasks where the user has to chose one of the options (hence the selection).

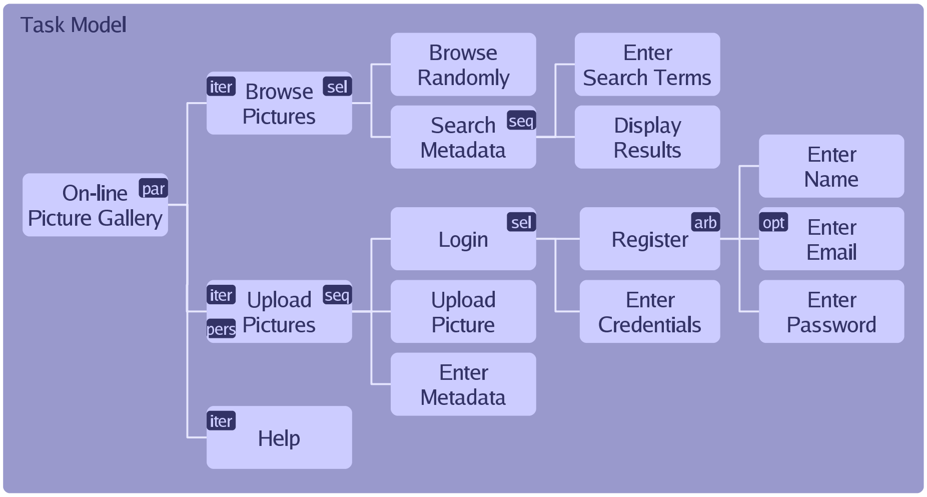

Fig. 2. Task Model

Browse Randomly is a leaf since the granularity of task refinement is adequately modelled for our purposes. *Search Metadata* is subdivided into two sequential subtasks, the first presenting a search form and the second displaying the results. The largest subtree of the application *Upload Pictures* relates to a process to register with the web application and to upload pictures. This process is marked as persistent; whenever a user leaves the web site before completing the process (subtree), it is persisted and presented again when the user returns. On the other hand, browsing pictures and invoking help are volatile, thus the states are discarded when the user leaves the web site. Uploading pictures consists of a task sequence: logging in, uploading a picture, and entering meta-data for the picture. *Login* is a selection between first time registration and authentication by using a given username/password combination. *Registration* is subdivided into three tasks, which can be performed in an arbitrary sequence: entering the user's name, entering an email address (which is optional) and choosing a password.

3 Domain Model

The domain model describes the objects of the business domain, their properties in terms of attributes, sub-object structures, and semantic relationships. Techniques adopted for defining this model correspond to well-known diagrams from Software Engineering (such as the class model in UML) or database engineering (ER diagrams). We will use UML (see figure 3). A simple entity class[1] *Account* is used to store user credentials and a service class[2] *AccountService* is used to create new accounts and to retrieve existing ones.

[1] See [1] for a definition of the stereotype *Entity*.
[2] See [1] for a definition of the stereotype *Service*.

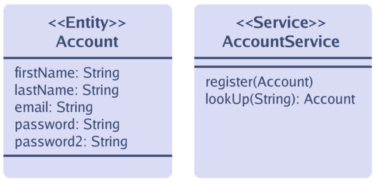

Fig. 3. Domain Model

4 Abstract Dialog Model

The task model is not yet suitable to generate the dialog between the user and the domain model. A dialog describes messages in terms of input and output interactions exchanged between the user and the system. From a users point of view, a dialog state represents the “position” within a dialog while the user is interacting with the application. The positions, i.e., the states, are changed by task execution. The tasks are associated with input and output declarations to specify what kind of data may be exchanged between the user and the system to perform a task. To enable this data exchange, appropriate interaction components have to be defined. Such information is typically described by means of a user interface model, comprising an abstract and concrete description of the presentation [20]. The abstract model describes *what* will be shown to the user at one point in time on a web page. The concrete model, also called the visual design, described *how* the web pages will be presented defining the concrete layout in terms of colors, fonts, the logo to be inserted and so on.

4.1 Basic Components and Grouping Mechanism

In our approach we introduce abstract dialog units (ADUs) for specifying the abstract user interface [2]. ADUs are associated to tasks from the task model, specifying abstract interface components for displaying output and transferring data from input elements. Thus, the abstract presentation of a web application is closely related to the dialog structure.

The abstract user interface can be constructed using predefined interaction objects, i.e., output objects (text, image) and input objects (text input, checkbox input). Since we are focusing on web applications, these components, which we refer to as generic interface components, correspond to HTML input and output elements. Figure 4 shows some of the components used in the example later on. *Label* just outputs some text, *Image* displays a picture. *Input* creates a text input and links it to a field from the domain model. The type attribute can automatically convert strings (HTTP request parameter are always strings) into domain model types and the validate attribute can perform simple constraint checks on user input (i.e. not empty, integer range, etc.). *Secret* has the same

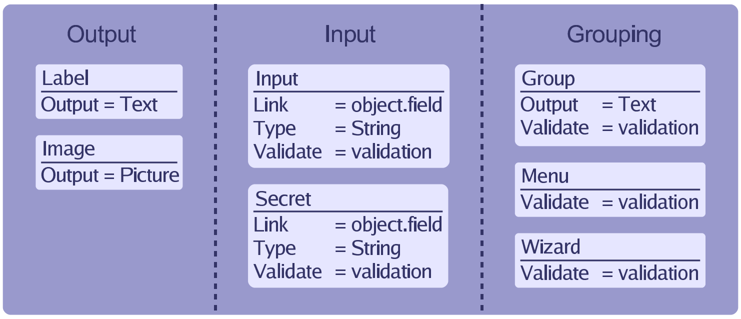

Fig. 4. Generic Interface Components

capabilities as an Input component, but the behavior of the input field displayed in the browser, which is cloaked with asterisks.

Since a task requires most of the time more than a single input or output, generic interface components can be grouped together using grouping components. A simple grouping component *Group*, specifies an ordered aggregation of its subcomponents. It outputs some text as the title of the group, displays the subcomponents as a list and can run validations on the aggregated subcomponents. *Menu* also groups subcomponents, but presents them as a link list to navigate to them individually, useful to create a menu. *Wizard* groups its subcomponents into sequential steps.

More complex components, like data grids and tree structures, can be added later on. The hierarchical structure of complex ADUs can be represented visually by means of a tree notation as shown in figure 5. The complex task *Enter Name* is composed of a set of label and input components, aggregated by the group component.

4.2 Connecting the Domain Model

The domain model can be created after the task model has been finalized, or, in case of legacy applications, can exist beforehand. In both cases, the domain model needs to be linked to the task model, and there are two steps involved.

Input components need to be linked to domain model fields, creating *value bindings*. These links are bidirectional, existing values from the domain are used to initialize the input fields. After a subsequent request arrives from the user, the user input is written back into the domain model.

Methods in the domain model can be invoked during task performance. Special fields in the ADUs called onEntry and onExit are used to link to methods in the domain model, creating *method bindings*. They are unidirectional, since method invocation always originates from the task model and targets the domain model.

While value bindings just transfer data from and to the domain model, method bindings can initiate processing on the data.

4.3 Abstract Dialog Model Example

The example (figure 5) in this section demonstrates the attachment of ADUs to tasks. First of all, the Register task is performed and the onEntry method binding is invoked, creating an empty Account object. The group component merely displays a title for all subtasks, the validation is later triggered, when the task *Register* is completed, i.e., all subtasks are completed (arbitrary sequence).

Enter Name creates two input fields with associated labels in a group and uses value bindings to link into the domain model. Validation just checks for empty strings.

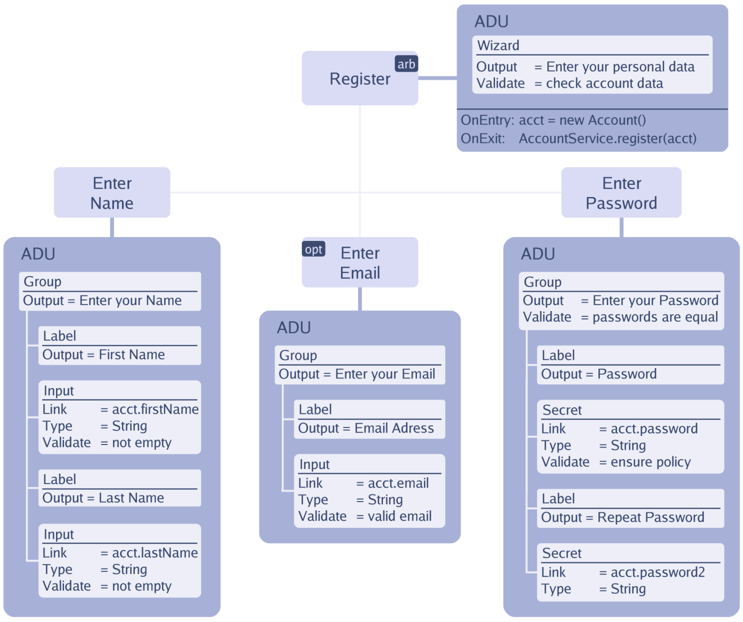

Fig. 5. Abstract Dialog Units Attached to Tasks

Enter Email creates a single input field and a label in a group. The validator checks for a syntactical correct email address and updates the value binding only if the email address string from the request qualifies.

Enter Password creates two input fields with asterisk cloaking and ensures for the first password, that it doesn't violate the password policy. The grouping component validates the entered passwords and checks if they are equal. Since the first password field doesn't know about the second password field, the check of equality can be performed at the grouping layer only, since here the data of both passwords entries are available to the validator.

After performing all three subtasks, the parent task *Register* is finished, the validation of the parent group is triggered, which checks the entered account data for inconsistencies created by contradicting inputs in separate subtasks. Finally, the onExit method binding is called and the account is created.

5 Runtime Architecture

Based on the models we described in the last sections, we propose a runtime architecture to process instances of these models, i.e. a task controller which processes task model instances and a dialog controller which processes abstract dialog model instances. Figure 6 displays the overview of the architecture and the flow of a request from a user's web client.

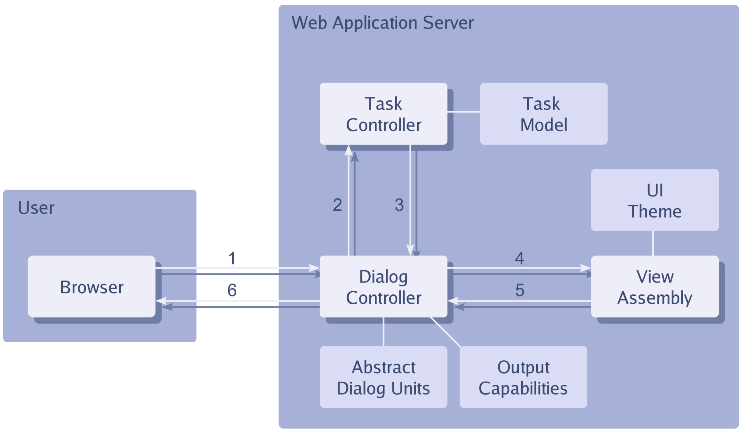

Fig. 6. Architecture View

1. All requests are handled by the *dialog controller*[3]. It associates requests to running task model instances and performs basic validation. **2.** The *dialog controller* invokes the *task controller*, which promotes the task model instance. **3.** The *task controller* returns. **4.** The *dialog controller* delegates the transforming of the ADU tree into a web page to the *view assembly*. **5.** The *view assembly* returns a web page. **6.** The *dialog controller* returns the web page to the browser.

The architecture is based on the classic model-view-controller (MVC) pattern for web applications [23], with a special focus on controllers. The *model* matches the domain model, the *view* is created in the view assembly and the *controller* corresponds to the task- and dialog controllers in cooperation with the task- and abstract dialog model.

Task- and dialog controller are explained in more detail in the next sections. The view assembly doesn't need a more detailed specification at the moment, because it just maps generic interface components to their HTML counterparts. Later on, with more complex generic interface components available, the mapping part will be more sophisticated.

[3] Since it acts as a single point of entry, it is also known as front controller.[1]

5.1 Task Controller

The task controller is responsible for creating task model instances from a dedicated task model and managing their lifecycles. For each task it creates a finite state machine (FSM, see figure 7), which consists of the possible task's states and the transitions between the states triggered by events.

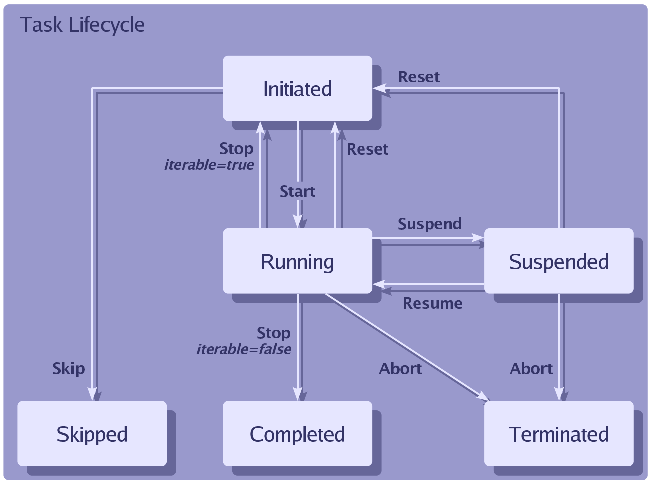

Fig. 7. Task Finite State Machine

Initially, the FSM is in the *initiated* state, and can be started or skipped. If skipped, the FSM changes into the *skipped* end state, counting the task as not completed but successfully finished. For example, if an optional task in a sequence is skipped, the subsequent task can be performed.

From *initiated*, the event start changes the FSM into the *running* state. By entering the *running* state, the method binding onEntry will be invoked. If the FSM receives a stop event, it will invoke the onExit method binding and change into the *completed* end state (or the *initiated* state, if the task is iterable).

Since we have to cope with suspensions, it is always possible that the task needs to be suspended while running. Depending on the lifespan of the task, it changes into the *suspended* state (if lifespan = suspendable or persistent) or in the *terminated* end state (if lifespan = volatile).

The hierarchy of tasks in the task model is adhered by adding conditions based on temporal relations to the transitions of the finite state machines. For example, to finalize a parent task (transition from *running* to *completed*) in a arbitrary sequence, all child tasks need to be in the state *completed* or *skipped*.

5.2 Dialog Controller

Fundamental part of the dialog controller is the *component tree*, which is used to interact with the user. ADUs are linked to tasks that need user interaction. Since we don't have generated pages yet, the information inherent to the task model hierarchy can be used to combine ADUs in order to present them on web pages. Using a task model instance from the task controller, the dialog controller creates a component tree by adding the ADUs from running tasks to a component tree, starting with the ADU from the root task and traversing the task hierarchy with a depth first search. The resulting "virtual" *generic interface component tree* was not modelled by the designer, but is created by the dialog controller to interact with the user. The component tree for the example in figure 5 is depicted in figure 8.

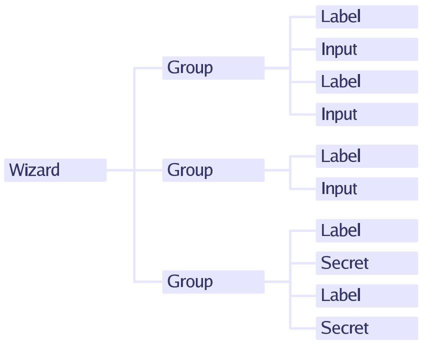

Fig. 8. Component Tree

For example, if a web site needs to be accessible using a desktop computer and a PDA, the dialog controller can decide at runtime how much screen size is used by the interface components and change the spatial appearance of grouping elements. The *Register* task in figure 5, could be presented on one page or on three subsequent pages. To combine all register tasks into a single page, the dialog controller would decide to change the appearance of the grouping element *Wizard* to display all its subcomponents at once. Since this can change the final user interface in a bad way (jumping recurring interface elements, no recognition of the web site), the change of grouping appearance should be applied to the component tree bottom-up and is still research in progress.

Figure 9 depicts the internal flow of the dialog controller, which is divided into six sequential phases[4].

Restore Component Tree. assigns a previously stored component tree from the user session to the request, which is only possible if the request originates from a web page created by the dialog controller (i.e., a component tree has

[4] This is inspired by the request processing of Java ServerFaces([19]).

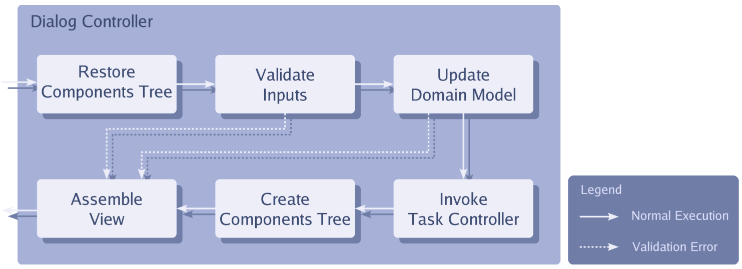

Fig. 9. Dialog Controller

been created and stored). If a component tree is found, the request attributes, which correspond to user input, populate the component tree and the next phases only use the component tree to process user interactions.

Validate Inputs. The validations of the generic interface components of the component tree are processed. If, for example, the last name field of the registration example (figure 5) has been left empty by the user, the validation would fail (mandates not empty) and the processing of the request would stop here and jump immediately to the Assemble View phase to create the same page with a validation error message, allowing the user to correct mistakes.

Update Domain Model. The values of the input components of the component tree are passed over to the domain. If the domain does not accept some of the input, the processing of the request would stop here again and jump immediately to the Assemble View phase.

Invoke Task Controller. The task controller creates a new instance of the task model if it doesn't exist and stores it in the user session. It promotes the task model instance by starting and stopping tasks depending on the users interactions. It invokes domain model methods by executing the onEntry and onExit method bindings of started and stopped tasks. The lifecycle of tasks is explained in section 5.1.

Create Component Tree. Due to the changes in the task model instance in the previous phase, the set of active tasks may have changed. The dialog controller creates a new component tree based on tasks in *running* state to interact with the user. After creating the tree, it is stored in the user session.

Assemble View. Pass the component tree to the view assembly, were a HTML page is created. Insert context information into the resulting page, which allows the dialog controller on a subsequent request to associate the user to a running task model instance.

6 Related Work

WebML [11] provides mainly a visual notation for defining data-driven hypertexts. The core is made up by a continually expanded set of so-called units, e.g., Data Unit, Entry Unit, Login Unit. The units represent what is to be included on a web page, whereby the expressiveness range from simple (e.g., a Data Unit represents information from the underlying data base) up to little processes (such as specified by the Login Unit). Application dependent sequences of task execution (called workflows) can be specified by means of operation units with additional information concerning control conditions. All in all, interaction elements, dialog control and presentation are not modelled separately.

Separation of concerns is better realized in UWE and WSDM. UWE [16] is an object-oriented approach supporting the whole life-cycle of web application development. UML is used, including some extension enabling the specification of, e.g., a navigation model. The approach has been extended for the purpose of modeling task sequencing (called business processes in UWE). Similarly, as in our approach, task modelling is distinguished within analysis and within design. At design level, a process (task) is modelled in terms of classes that are refined by UML activity diagrams. A process class specifies the information needed for performance, including state information for the purpose of handling interruptions. The integration of the process classes into the navigation model result into two separated but connected spaces: one dedicated to process performance and the other dedicated to "pure" navigation / browsing. In our approach, similar to OO-H [9], both aspects are interwoven in the navigation model.

The the objects chunks of WSDM [14] are similarly to the ADUs proposed by our approach. While in WSDM such chunks are only attached to leaf tasks, in the work presented here ADUs can be attached to a task regardless of its position within the task hierarchy. By this, we can easily denote aspects being relevant to a complete sub-task tree.

6.1 Current State of Work

Our work differs from all mentioned approaches above with respect of utilizing the “task views”. In most approaches, including those developed in the field of HCI, e.g., [20], [12], [18] the task / process structure is transformed into an initial dialog model or navigation model at design time. We are postponing dialog creation at runtime to adapt the user interface to client specific requirements. The biggest drawback of this approach is poor imagination of the generated web pages by the task designer, which we try to counter with predictable behavior of grouping elements. Still, there is a trade-off between adaptability and user interface consistency.

The concepts were applied in some projects so far. For this purpose, our previous work on simulating task models ([4]) was extended to support interactive web applications at run time (an example of which was presented in ([3]). As shown in ([5], [6]) the web-MVC pattern was extended by task-related components. A task processing manager is responsible for keeping track of state information at the level of task and process execution, respectively.

We are implementing task- and dialog controllers to support our ideas of task suspension and runtime dialog generation. We have developed an XML schema of our task notation, to support easy integration of task models into the controllers. The task controller, which creates FSMs of the tasks is finished and deployable. The dialog controller, which uses a component tree and generates web pages is also in a running state. Current work aims at the integration of both controllers into a coherent runtime system. An editor to create task models has been developed and is able to export the models in XML.

References

1. Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns: Best Practices and Design Strategies, 1st edn. Prentice Hall, Englewood Cliffs (2001)
2. Betermieux, S., Bomsdorf, B., Langer, P.: Towards a generic model for specifying different views on the dialog of web applications. In: Proceedings of HCI International. Lawrence Erlbaum Associates, Mahwah, NJ (2005)
3. Biedebach, A., Bomsdorf, B., Schlageter, G.: The changing role of instructors. In: Proceedings eLearn 2002 (2002)
4. Biere, M., Bomsdorf, B., Szwillus, G.: The visual task model builder. In: Proceedings of CADUI 1999 (1999)
5. Bomsdorf, B.: First steps towards task-related web user interfaces. In: Proceedings of Computer-Aided Design of User Interfaces, pp. 349–356 (2002)
6. Bomsdorf, B.: Task modeling for customization of web applications. In: Proceedings HCI International 2003, pp. 33–37 (2003)
7. Bomsdorf, B., Szwillus, G.: Towards a universal modelling tool. In: Palanque, P., Paternó, F. (eds.) DSV-IS 2000. LNCS, vol. 1946, Springer, Heidelberg (2001)
8. Brambilla, M., Ceri, S., Fraternali, P., Manolescu, I.: Process modeling in web applications. In: ACM Transactions on Software Engineering and Methodology (2006)
9. Cachero, C., Gómez, J., Pastor, O.: Object-oriented conceptual modeling of web application interfaces: the OO-H method. In: ECWEB'00, pp. 206–215. Springer-Verlag, Heidelberg (2000)
10. Cachero, C., Koch, N.: Conceptual navigation analysis: a device and platform independent navigation specification. In: Proceedings of 2nd International Workshop on Web-Oriented Software Technology (2002)
11. Ceri, S., Fraternali, P., Matera, M., Maurino, A.: Designing multi-role, collaborative web sites with WebML: a conference management system case study. In: IWWOST 2001 Workshop (2001)
12. Clerckx, T., Luyten, K., Coninx, K.: The mapping problem back and forth: customizing dynamic models while preserving consistency. In: TAMODIA '04. Proceedings of the 3rd annual conference on Task models and diagrams, pp. 33–42. ACM Press, New York, NY, USA (2004)
13. de Troyer, O.: Audience-driven web design. In: Information modelling in the new millennium, IDEA Group Publishing (2001)
14. de Troyer, O., Casteleyn, S.: Modeling complex processes for web applications using WSDM. In: Proceedings of the Third International Workshop on Web-Oriented Software Technologies, IWWOST (2003)
15. Isakowitz, T., Kamis, A., Koufaris, M.: The extended RMM methodology for web publishing. Working Paper IS-98-18 (1998)
16. Koch, N., Kraus, A., Cachero, C., Meliá, S.: Integration of business processes in web application models. Journal of Web Engineering 3(1), 22–49 (2004)
17. Kraus, A., Koch, N.: A metamodel for UWE. Technical report, University of Munich (2003)

18. Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J.: Derivation of a dialog model from a task model by activity chain extraction. In: DSV-IS, pp. 203–217 (2003)
19. McClanahan, C., Burns, E., Kitain, R.: JavaServer faces specification, v1.1, rev. 01 (2004)
20. Paternò., F., Mancini, C., Meniconi, S.: Concurtasktrees: A diagrammatic notation for specifying task models. In: INTERACT '97. Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction, London, UK, pp. 362–369. Chapman & Hall, Ltd., Sydney (1997)
21. Puerta, A., Cheng, E., Ou, T., Min, J.: MOBILE: User-centered interface building. In: Proceedings of CHI 99. ACM Press, New York, NY, USA (1999)
22. Rossi, G., Schwabe, D., Lyardet, F.: Web application models are more than conceptual models. In: Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., Métais, E. (eds.) ER 1999. LNCS, vol. 1728, Springer, Heidelberg (1999)
23. Singh, I., Stearns, B., Johnson, M.: Designing Enterprise Applications with the J2EE Platform, 2nd edn. Addison-Wesley, London, UK (2002)
24. Stary, C.: Task- and model-based development of interactive software. In: Proceedings of IFIP 98 (1998)
25. van der Veer, G.C., Lenting, B.F., Bergevoet, B.A.J.: Groupware task analysis - modelling complexity. Acta Psychologica (1996)
26. Vilain, P., Schwabe, D.: Improving the web application design process with UIDs. In: Proceedings of 2nd International Workshop on Web-Oriented Software Technology (2002)