

Model Transformations for Integrating and Validating Web Application Models

Alexander Knapp, Gefei Zhang*

Ludwig-Maximilians-Universität München
{knapp, zhang}@pst.ifi.lmu.de

Abstract: While most current Web Engineering methodologies model the separate aspects, content, navigation, business logic, and presentation, of Web systems in separate models, integration of the different models and in particular the validation of their interaction is not yet sufficiently supported. We propose a systematic approach of building a UML state machine that integrates the separate concerns content, navigation, and business logic of a Web system into a big picture, which can then be validated formally for consistency and behavioural properties.

1 Introduction

Separation of concerns is an important principle of software system modelling, which is applied in almost every software design. The advantage of a clean separation of concerns includes better readability and enhanced reusability of software models. On the other hand, the models may be separate, but not isolated: they must be consistent to each other and their interaction must specify the system as a whole.

Most current Web Engineering methodologies (for an overview see [Sch01]) follow the principle of separation of concerns and model the different aspects of a Web application in separate models: the content, the navigation structure, the business process logic, and the presentation. However, integration and validation of these separate models, in particular the validation of their interaction, is not yet sufficiently supported.

We propose, in the context of UML-based Web Engineering (UWE [KK03]), a systematic approach of merging the separate models of a Web application and building an integrated big picture by model transformation. The model transformations are given as graph transformation rules on the UWE metamodel. The result of the transformation is a UML [Obj05] state machine, which includes the static navigation structure as well as the dynamic behaviour of a Web application model. The state machine can then be validated formally using, e.g., model checking.

Two features make UWE particularly amenable for our integration and validation approach: UWE is defined as a UML profile using a conservative extension of the UML

*Partially supported by the DFG project MAEWA (WI 841/7-1).

metamodel [KK03], therefore we can apply the well-known technique of metamodel-based graph transformations (cf., e.g., [BH02]). Furthermore, UWE includes support for modelling complex business processes using UML activity diagrams (see [KKZH04], where also a comparison with other Web engineering approaches can be found); thus the behavioural properties of the overall system are a primary concern.

Using UML state machines as the results of the integration process is motivated by the possibility of applying formal techniques for validation, like model checking [KMR02], directly to a familiar software design notation. Moreover, the expressiveness of UML state machines offers ample opportunity for refining the integrated model of a Web application and including features like access control [ZBKK05].

The remainder of this paper is structured as follows: In Sect. 2 we give an overview of the UWE method and notation, and introduce a running example of a music portal. The model transformations for integrating the different Web engineering models of the music portal into a UML state machine are described in Sect. 3. In Sect. 4 we demonstrate the validation and refinement of the integrated state machine. Related work is discussed in Sect. 5. We conclude with some remarks on future work.

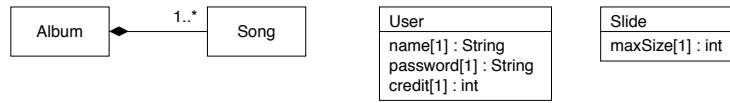
2 UML-based Web Engineering

UML-based Web Engineering (UWE) models a Web application from different points of view: the content, the navigation structure, the business processes, and the presentation. UWE is defined in the form of a UML profile and an extension of the UML metamodel (for more details see [KK03]), and is therefore UML-compliant.

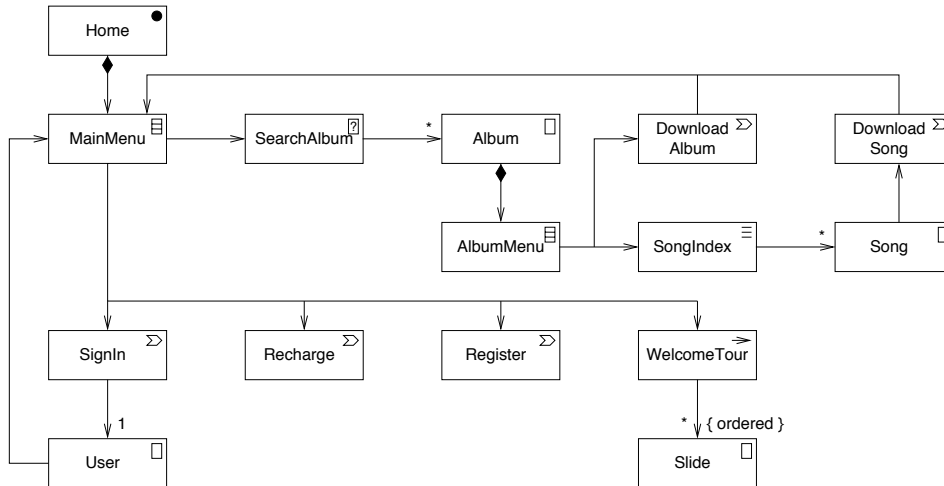
We recall UWE by means of a simple music portal example, inspired by `www.mp3.com`, which provides albums and songs for download: Users can search for albums and choose to download a complete album, or, after selecting a song from an album to download this song. Downloading is reserved for members only and users have to sign-in before download is available. Users can register and become members of the portal. Each album and each song costs some amount of money, and downloading is only possible as long as the user has enough credit. A member can also recharge his credit.

In UWE, the content of Web applications is modelled in a content model where the classes of the entities that will be used in the Web application are represented by instances of the UML metaclass `Class`. Relationships between contents are modelled by UML associations. The content model of the simple music portal is given in Fig. 1(a); additionally to providing albums and songs and managing users, the music portal also has an introductory presentation of its main features in slides.

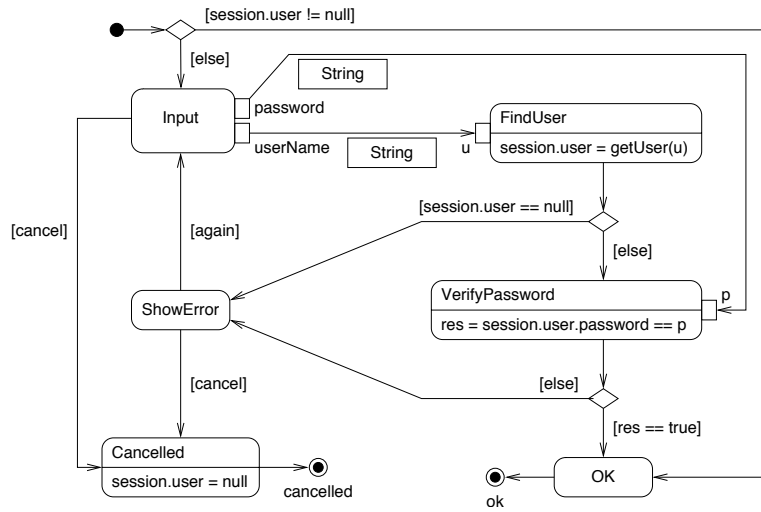
Based on the content model, the navigation structure of a Web application is laid down in an UWE navigation model; for our music portal example this model is shown in Fig. 1(b). Navigable nodes are represented by instances of the metaclass `NavigationNode`, which is a subclass of `Class`. Instances of Association-subclass `Link` model direct links between two navigation nodes. There are several subtypes of `NavigationNode`: `NavigationClasses` represent contents (visualised by \square), like `Album` or `Song`; these are derived from the con-



(a) Content model







(b) Navigation model








(c) Business process SignIn

Figure 1: UWE model of the music portal example

tents model. Navigation paths are structured by instances of Menu (≡), Index (≡), Query

() and Guided Tour (). Menus (like AlbumMenu) represent choices between different subsequent navigation nodes. In contrast, indexes (like SongIndex) have a single successor node, the user selects a particular instance of this successor navigation node. A query (like SearchAlbum) models a search action of the Web application, where a user can enter a term and select from the matching results. Guided tours (like WelcomeTour) represent the entry point to an ordered set of navigation class instances, where the user may go forward or backward. Moreover, exactly one node is marked as the starting point of the Web application (Home, ), by setting the attribute isHome of NavigationNode to true. Business processes, which are modelled in UML activity diagrams, are integrated into the navigation model by instances of ProcessNode (), another subclass of NavigationNode.

The navigation structure of the music portal example (Fig. 1(b)) is as follows: node Home is the starting point of the application, from MainMenu a user can learn how to use this portal in a guided tour, or search for a certain album and then download the complete album and/or its songs, or—after signing in as a member—view his personal info. The business processes Recharge and Register are also reachable from the main menu.

The activity diagrams modelling the business processes are based on the UML 2.0 specification [Obj05]: Actions (like Input, ) model the actions a user and the system must carry out to complete a business process, Pins (like password, ) represent data flow between the actions. The flow of actions is started in the initial node () and terminates in the final node (); branching is introduced by decision nodes ().

In Fig. 1(c) the business process SignIn is modelled: if the user of the current session is not yet signed in, he is asked to input his user name and password, both of type String. If the given username and password are invalid, then an error page is shown, otherwise it is noted in the session that the user has signed in. The same process is used in DownloadSong and DownloadAlbum (not shown here) to ensure that the user is signed in before download; furthermore, these business processes offer the possibility to recharge the user's account without going through the main menu.

The desirable properties of this model of a music portal include the possibility of reaching the download page of an album or a song, once signed in; that it is only necessary to sign in once in order to download several songs; or that the correct amount of money is billed for downloading. In fact, the first two properties involve both the navigation structure and the business logic and thus rely on the consistent interaction of different models.

3 Model Transformations for Integration

We combine into a UML state machine the navigation structure and the business processes of a Web application modelled in UWE. First, we transform the navigation model into a basic state machine, then we integrate the business processes into this state machine as submachine states. The transformation steps are given by graph transformation rules on an extension of the UWE metamodel: Each rule consists of a left-hand side pattern graph, an optional negative application condition, and a right-hand side graph. Model elements and their connections that match the left-hand side pattern of a rule and do not match

the negative application condition are replaced by the graph on the right-hand side; the negative application condition is depicted by crossed-out graph structures in the left-hand side pattern. The graph transformation rules are defined on an extended metamodel such that information can be stored during processing. In order to keep the presentation simple, we forgo to define all rules in detail and will describe some transformations pictorially.

UML 2.0 state machines [Obj05] consist of states (\square) and transitions between states. States may be simple or composite, where composite states contain at least one so-called region which again contains states. Transitions may either directly connect states or start from exit points (\otimes) and end in entry points (\circ) for avoiding transitions crossing regions and thus fostering encapsulation of states. Each state can show an entry action that is executed on entering the state. Transitions can be labelled by a triggering event, a guard, and effect actions. A transition is enabled if its source state is currently active, its triggering event is present, and its guard evaluates to true; on firing a transition its effect is executed.

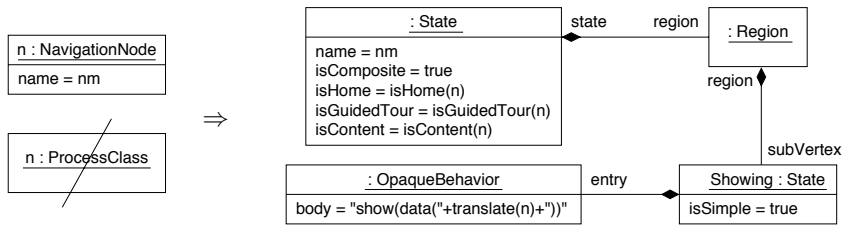
A UML state machine is executed in a context that allows to store and retrieve global information. For our integrated state machine we assume a session context that holds the currently signed-in user (there may be none), and currently selected content entities, like the song to download.

Roughly speaking, our model integration transformations render each UWE navigation node, including navigation classes and process classes, as a composite state with as many entry and exit points as there are links leading to and leaving from the navigation node, respectively. The data flow is taken into account, on the one hand, by passing parameters to the composite states that represent navigation classes; and, on the other hand, by storing navigation information in the session context.

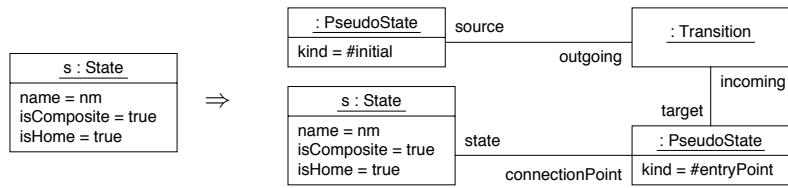
3.1 Transforming navigation structures

The navigation model is transformed into a basic state machine by the following rules: For every navigation node, which is not a process class, we create a composite state with the same name as the navigation node. This composite state shows exactly one region which contains a simple state *Showing* (see Fig. 2(a)). For a home navigation node, we additionally create an initial state and an entry point for its composite state and connect the initial state and the entry point by a transition (Fig. 2(b)). Whether the navigation node is a guided tour or a navigation class is recorded in the attributes *isGuidedTour* and *isContent* for later reference; these attributes are not part of the UWE metamodel. For every process node a composite state with no substates is created (rule not shown here). Note that at this interstage we also accept models that have states connected to each other by links.

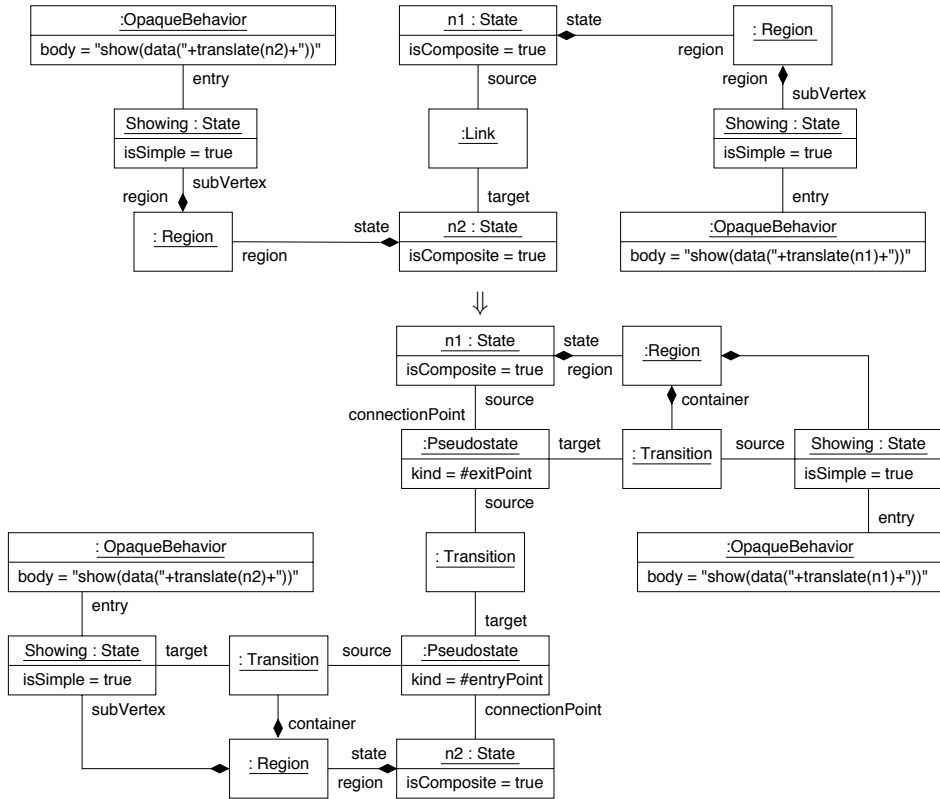
In the next step, the links between the states are transformed into transitions (Fig. 2(c)). A transition leaves its source state at an exit point and leads to an entry point of the subsequent state. We also create a trigger with a parameter for those transitions that lead to a state representing a navigation class (like *Album* or *Song*) in order to indicate which instance of the navigation class is to be shown to the user. Such a parameter is not created if the original link has a multiplicity of one at the target end (like *User*), since this means



(a) Mapping navigation nodes



(b) Mapping the home node



(c) Mapping links to transitions

Figure 2: Rules for mapping nodes and links to states and transitions

that the user has no choice for this navigation class and thus a parameter is not necessary.

Each Showing state has an entry action `show` that transforms information, gathered by another function `data`, to HTML or PDF or any “downloadable” format. The function `data` receives a parameter that indicates the navigation node instance for which data must be gathered and depends on the kind of navigation node which the state represents. Navigation classes refer to the content model of the Web system and thus may have more than one instance. Therefore, the parameter of `data` must specify the instance to show and `data` will retrieve its contents. Furthermore, in this case, the composite state containing Showing also stores by its entry action this instance in a variable in the session such that subsequent states can refer to this information. For the other navigation nodes it suffices to give `data` the name of the navigation node. Which parameter `data` receives, is decided by the function `translate`. The result of transforming navigation class `Song` is depicted in Fig. 3(a). For queries `data` returns the fields needed for the user input; for menus, indexes and guided tours it returns the links to present to the user.

Finally, every state preceded by an `isGuidedTour` state is enhanced by two transitions from Showing to itself to represent the user navigating to the previous and the next step in the guided tour, respectively; see Fig. 3(b).

The resulting top-level state machine for the music portal example is shown in Fig. 3(c). We have also indicated the parameters transferred to the states representing navigation classes.

Whole/part relations represented by composite aggregations in the navigation model may be emphasised in the state machine. Alternatively to as described above, the composite navigation node may be mapped to a composite state containing several regions: one for its own Showing state, and one for each of its parts, containing a substate representing the part. Figure 4 shows how `MainMenu` is integrated into `Home` using this alternative rule.

3.2 Integrating Business Processes

After creating from the navigation model the top-level states and transitions between them, we now integrate the activity diagrams modelling the business processes as substates. Initial nodes, final nodes, decision nodes, action nodes, and control flow edges are transformed into their counterparts in UML state machines. In contrast, data flow is not directly supported by state machines and has to be simulated.

Figure 5 summarises the rules informally: for the initial node we create an entry point of the parent (top-level) state, each final node is transformed into an exit point with the same name. Decision nodes are mapped to junctions. Each action node is mapped to a simple composite state with the same name and an entry action which performs the task of the original action. Data flow from action node A to action node B using the pins `p` and `q` is simulated by first turning `q` into a global variable in the state and then defining an exit action of the state A that assigns `p` to `q`.

Note that these rules are by no means complete for transforming UML 2.0 activity dia-

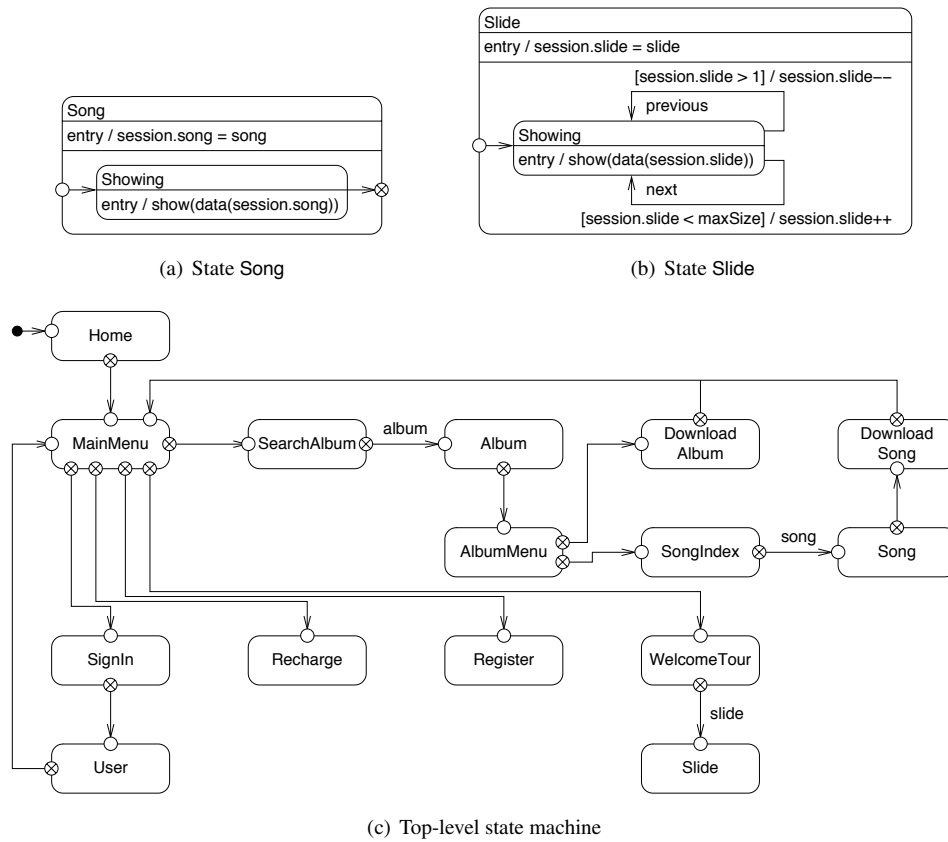


Figure 3: State machine for music shop example

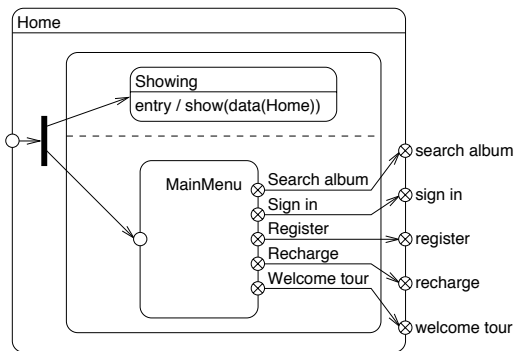


Figure 4: Composite state Home

grams, but suffice for UWE business processes. Figure 6 displays the result of the transformation rules applied to the SignIn business process (see Fig. 1(c)).

4 Model Validation

The integrated state machine model of an UWE Web application model is the basis for validating the design. In the following, we use the UML model transformation tool Hugo/RT [KMR02]¹ for applying model checking to the UML state machine. Hugo/RT translates state machines, collaborations, and assertions into input models for the model checkers SPIN and, for real-time state machines, UPPAAL².

We use the linear-time temporal, on-the-fly model checker SPIN as the backend for the translation with Hugo/RT. As a first check of our integrated model, we require that it is possible to download a song:

```
assertion reachability {
    F system.inState(DownloadSong.Downloading);
}
```

Figure 7 shows the composite state DownloadSong in the integrated state machine. Although a simple property, its verification requires to consider both the navigation structure and the business processes.

For performing actual model checking we have to abstract from some details of the model: On the one hand, user interaction (like cancelling or retrying the sign-in process) is replaced by non-deterministic choice; on the other hand, we do not include concrete data for albums or songs, but content ourselves with a single item of each. Employing this abstraction, the reachability assertion, saying that there is a path such that sometime (F) the state machine (**system**) is in state DownloadSong.Downloading, is indeed verified by SPIN and the witnessing example provided by SPIN can be retranslated, using again Hugo/RT, into a UML run showing how to reach DownloadSong.Downloading.

¹<http://www.pst.ifi.lmu.de/projekte/hugo>

²<http://spinroot.com>, <http://www.uppaal.com>

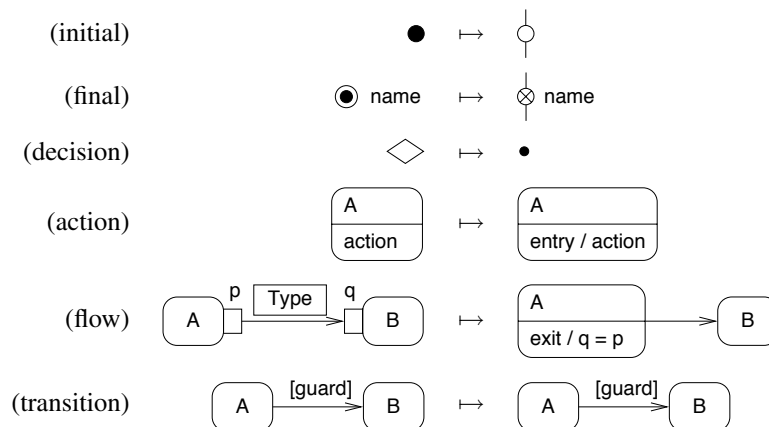


Figure 5: Informal rules for translating business processes into a state machine

Finally, we check that it is impossible to download songs forever, without recharging one's account:

```
assertion download {
  not ((G F system.inState(DownloadSong.Downloading)) implies
    (G F (system.inState(Recharge) or
      system.inState(DownloadAlbum.Recharge) or
      system.inState(DownloadSong.Recharge))));
}
```

Again, after translation with Hugo/RT, SPIN answers that this is impossible. However, we have not checked that each downloaded song is billed correctly (where we assume that each song costs a unit of money): Instrumenting the state machine with an auxiliary attribute recording the user's credit before downloading a song and asserting in Downloading that the current credit is smaller than the credit before download:

```
simple Downloading {
  entry assert(creditBefore < credit);
}
```

reveals an erroneous transition in DownloadSong: After recharging, the downloaded song is not billed. In order to correct this, the effect `session.user.credit--` has to be moved below the junction state before Downloading.³

Beyond the ability to check whether the UWE Web application models interact consistently with each other, we may also use the integrated state machine as the starting point for refinements that are not easily expressed in UWE. For example, even though access to node User is explicitly guarded by the SignIn process in the navigation model, it may be possible to access User directly by an external link [ZBKK05]. Such misuse can be avoided by refining the composite state for User by a check for `session.user != null`, like in the composite state for DownloadSong.

5 Related Work

Our approach uses graph transformation systems to integrate the navigation structures and the business processes of a UWE model of a Web application into a UML state machine. This is, as far as we know, the first method of systematically combining the different models of a Web system into a big picture and then verifying the correctness of the models and their interaction.

Other Web Engineering methods such as OOHDM [SH04], OO-H [KKCM04], WSDM [Tro03], or WebML [CDDF05] provide no integration of business process models that are as expressive as UWE's: they integrate only entry and exit points of the actions composing the business processes into the navigation structure, but not the actions themselves.

Navon and Bustos [NB05] as well as Dolog [Dol04] also use state machines to model Web systems. The latter approach also affords the integration of WebML rules that change the

³The integrated UML state machine is available at <http://www.pst.ifi.lmu.de/projekte/hugo/uwe.ute>.

navigation structure of an adaptive Web system [CDMN05]. In comparison, not only is our big picture state machine more expressive by representing data flow, but our approach also gives a concrete method for creating the big picture.

W2000 [BCM05] is, to our knowledge, the only method that enforces model consistency, although the method does not support business process modelling and thus only consistency between the navigation model and the content model is considered.

Verifying models of Web systems is still in its infancy. Chen and Zhao [CZ04] outline how to use labelled transition systems to verify the behaviour of a client with caching. Model checking is applied in [SDM⁺05] to verify the navigation structures of Web application models. To the authors' knowledge, our approach is novel in also model checking dynamic business processes in Web systems.

6 Conclusions and Future Work

We presented a systematic approach of integrating the navigation structure and the business processes into a state machine and how to use this state machine to verify the interaction of the different models specifying a Web system by model checking. Thanks to the formal rules formulated as a graph transformation system, the integration can be automated. We plan to extend ArgoUWE [KKZH04], the CASE tool of UWE, to generate the big picture automatically.

While model checking is a fast and simple way of checking whether a property holds or not, its main disadvantage is that the complete state space must be verified to find a counter-example. Not only is verification of properties in unlimited state space (e.g., in case of an unbound integer value) often impossible but also finite state systems may be problematic due to state explosion. Therefore we plan to apply other technologies such as automatic abstraction or interactive verification to Web models. Moreover, in situations where correctness of the models cannot be fully proved by formal methods, it is interesting to use techniques of testing UML state machines to get some help in finding possible design deficiencies.

In terms of Model Driven Architecture (MDA [KWB03]), our proposal provides a transformation from a platform-independent model (PIM) to another. The resulting big picture may be used as a starting point for further transformations. In particular, the WebSA approach [MCG03] may be applied for performing PIM to PSM transformation.

References

- [BCM05] Luciano Baresi, Sebastiano Colazzo, and Luca Mainetti. First Experiences on Constraining Consistency and Adaptivity of W2000 Models. In Hisham Haddad, Lorie M. Liebrock, Andrea Omicini, and Roger L. Wainwright, editors, *Proc. 2005 ACM Symp. Applied Computing (SAC'05)*, pages 1674–1678, 2005.

- [BH02] Luciano Baresi and Reiko Heckel. Tutorial Introduction to Graph Transformation: A Software Engineering Perspective. In Andrea Corradini, Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Proc. 1st Int. Conf. Graph Transformation*, volume 2505 of *Lect. Notes Comp. Sci.*, pages 402–429. Springer, Berlin, 2002.
- [CDDF05] Stefano Ceri, Florian Daniel, Vera Demaldé, and Federico Michele Facca. An Approach to User-Behavior-Aware Web Applications. In David Lowe and Martin Gaedke, editors, *Proc. 5th Int. Conf. Web Engineering(ICWE'05)*, volume 3579 of *Lect. Notes Comp. Sci.*, pages 417–428. Springer, Berlin, 2005.
- [CDMN05] Stefano Ceri, Peter Dolog, Maristella Matera, and Wolfgang Nejdl. Adding Client-Side Adaptation to the Conceptual Design of e-Learning Web Applications. *J. Web Engineering*, 4(1):21–37, 2005.
- [CZ04] Jessica Chen and Xiaoshan Zhao. Formal Models for Web Navigations with Session Control and Browser Cache. In Jim Davies, Wolfram Schulte, and Michael Barnett, editors, *Proc. 6th Int. Conf. Formal Methods and Software Engineering (ICFEM'04)*, volume 3308 of *Lect. Notes Comp. Sci.*, pages 46–60. Springer, Berlin, 2004.
- [Dol04] Peter Dolog. Model-Driven Navigation Design for Semantic Web Applications with the UML-Guide. In Maristella Matera and Sara Comai, editors, *Engineering Advanced Web Applications — Proc. Wshs. conn. 4th Int. Conf. Web Engineering*, pages 75–86. Rinton Press, 2004.
- [KK03] Nora Koch and Andreas Kraus. Towards a Common Metamodel for the Development of Web Applications. In Juan Manuel Cueva Lovelle, Bernardo Martín González Rodríguez, Luis Joyanes Aguilar, José Emilio Labra Gayo, and María del Puerto Paule Ruíz, editors, *Proc. 3rd Int. Conf. Web Engineering*, volume 2722 of *Lect. Notes Comp. Sci.*, pages 497–506. Springer, Berlin, 2003.
- [KKCM04] Nora Koch, Andreas Kraus, Cristina Cachero, and Santiago Meliá. Integration of Business Processes in Web Application Models. *J. Web Eng.*, 3(1):22–49, 2004.
- [KKZH04] Alexander Knapp, Nora Koch, Gefei Zhang, and Hanns-Martin Hassler. Modeling Business Processes in Web Applications with ArgoUWE. In Thomas Baar, Alfred Strohmeier, Ana Moreira, and Stephen J. Mellor, editors, *Proc. 7th Int. Conf. Unified Modeling Language (UML'04)*, volume 3273 of *Lect. Notes Comp. Sci.*, pages 69–83. Springer, Berlin, 2004.
- [KMR02] Alexander Knapp, Stephan Merz, and Christopher Rauh. Model Checking Timed UML State Machines and Collaborations. In Werner Damm and Ernst Rüdiger Olderog, editors, *Proc. 7th Int. Symp. Formal Techniques in Real-Time and Fault Tolerant Systems*, volume 2469 of *Lect. Notes Comp. Sci.*, pages 395–416. Springer, Berlin, 2002.
- [KWB03] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.
- [MCG03] Santiago Meliá, Cristina Cachero, and Jaime Gomez. Using MDA in Web Software Architectures. In *Proc. 2nd OOPSLA Wsh. Generative Techniques in the Context of Model Driven Architecture*, Anaheim, 2003.
- [NB05] Jaime Navón and Pablo Bustos. Web Application Development: Java, .Net and Lamp at the Same Time. In David Lowe and Martin Gaedke, editors, *Proc. 5th Int. Conf. Web Engineering (ICWE'05)*, volume 3579 of *Lect. Notes Comp. Sci.*, pages 185–190. Springer, Berlin, 2005.

- [Obj05] Object Management Group. Unified Modeling Language: Superstructure, version 2.0. Specification, OMG, 2005. <http://www.omg.org/cgi-bin/doc?formal/05-07-04>.
- [Sch01] Daniel Schwabe, editor. *Proc. 1st Int. Wsh. Web-Oriented Software Technology (IWWOST'01)*, Valencia, 2001. <http://www.dsic.upv.es/~west2001/iwwost01/>.
- [SDM⁺05] Eugenio Di Sciascio, Francesco M. Donini, Marina Mongiello, Rodolfo Totaro, and Daniela Castelluccia. Design Verification of Web Applications Using Symbolic Model Checking. In David Lowe and Martin Gaedke, editors, *Proc. 5th Int. Conf. Web Engineering (ICWE'05)*, volume 3579 of *Lect. Notes Comp. Sci.*, pages 69–74. Springer, Berlin, 2005.
- [SH04] Hans Albrecht Schmid and Oliver Herfort. A Behavioral Semantics of OOHDMM Core Features and of Its Business Process Extension. In Nora Koch, Piero Fraternali, and Martin Wirsing, editors, *Proc. 4th Int. Conf. Web Engineering (ICWE'04)*, volume 3140 of *Lect. Notes Comp. Sci.*, pages 74–87. Springer, Berlin, 2004.
- [Tro03] Olga De Troyer. Modeling Complex Processes for Web Applications using WSDM. In Daniel Schwabe, Oscar Pastor, Gustavo Rossi, and Luis Olsina, editors, *Proc. 3rd Int. Wsh. Web-Oriented Software Technologies (IWWOST'03)*, Oviedo, 2003.
- [ZBKK05] Gefei Zhang, Hubert Baumeister, Nora Koch, and Alexander Knapp. Aspect-Oriented Modeling of Access Control in Web Applications. In *6th Int. Wsh. Aspect Oriented Modeling (AOM'05)*, Chicago, 2005.