Aspect-Oriented Modeling of Web Applications with HiLA

Gefei Zhang[1] and Matthias Hölzl[2,⋆]

[1] Arvato Systems Technologies GmbH
[2] Ludwig-Maximilians-Universität München
{gefei.zhang, matthias.hoelzl}@pst.ifi.lmu.de

Abstract. Modern web applications often contain features, such as landmarks, access control, or adaptation, that are difficult to model modularly with existing Model-Driven Web Engineering approaches. We show how HiLA, an aspect-oriented extensions for UML state machines, can represent these kinds of features as aspects. The resulting models achieve separation of concerns and satisfy the "Don't Repeat Yourself" (DRY) guideline. Furthermore, HiLA provides means to detect potential interferences between features and a declarative way to specify the behavior of such feature combinations.

1 Introduction

The history of Model-Driven Web Engineering (MDWE) is also a history of Separation of Concerns. Even in the early hours of MDWE, numerous modeling approaches, such as [3,8,10,18], which considered only the static web sites with primitive GUIs common at that time, were designed so that the domain model, the navigation model, and the presentation model, were separated from each other.[1] This way, the complexity of the models could be reduced, the legibility and maintainability improved.

While web applications evolved to modern, ubiquitous, adaptive applications implementing complex business processes presented by elaborate GUIs, new concerns also emerged and had to be taken into account. Unfortunately, models of these concerns are often tightly entangled with the rest of the application and therefore hard to separate. For instance, adapting the behavior of a web application to different navigation patterns of different users often means introducing changes throughout the model so that the adaptation mechanism is interwoven with the normal application behavior. This makes the effect of adaptation difficult to discern in the model and, even more importantly, makes it difficult to consistently modify the adaptive behavior. Similarly, access control in the context of web applications often requires checking the current user's rights throughout in the navigation structure.

The growing complexity of web applications also poses another challenge for the separation-of-concerns efforts of the MDWE research: the growing number of concerns

⋆ This work has been partially sponsored by the EU project ASCENS, 257414.

[1] There were also approaches that did not care about separation of concerns, though. In these approaches the model just contained everything, i.e. navigation, presentation, etc. However, we think a clean separation of concerns is generally beneficial w.r.t. model readability. See also [11].

in web applications increases the chance that some of them are interfering. Modeling the interaction of concerns, i.e. how concerns are combined with each other, is often quite unintuitive, changing the interaction logic an error-prone task.

It is therefore desirable to have a language which supports 1) the separate modeling of different concerns of web applications, and 2) a high-level, i.e. declarative definition of the combination of concerns. In this paper, we present the power of the language *High-Level Aspects* (HiLA) in Model-Driven Web Engineering. HiLA is an aspect-oriented extension of UML state machines [15] and can be used on top of state-machine-based MDWE approaches, such as [2,6,14,20]. In HiLA, different concerns of a software system are modeled in *aspects*, separately from the base functionalities of the applications and from each other. Therefore, different behavioral concerns of a web application can be cleanly separated. Moreover, HiLA is defined in such a way that potential interference between aspects can be detected mechanically, and that combination of aspects can be defined in a simple, declarative way. Hence, HiLA can be very useful in model-driven engineering of modern web applications. HiLA is integrated in the Hugo/RT UML model translator which supports formal software-engineering aproaches with model checking, theorem proving, and code generation for HiLA models.

The remainder of this paper is organized as follows: in the following Sect. 2 we briefly overview the techniques of modeling web applications using UML state machines and point out some modularity problems. After a short introduction of the HiLA language in Sect. 3 it is shown in Sect. 4 how HiLA can help mitigate the problems. Combination of concerns is discussed in Sect. 5. Related work is discussed in Sect. 6 before we finally draw conclusions and sketch some future work in Sect. 7.

2 Modeling Web Applications with UML State Machines

The language of UML state machines is very popular for behavioral modeling. It is even considered “the most popular modeling language for reactive components” [7]. Therefore, it also provides a natural and widely-used way of modeling web applications, see e.g. [2,6,13,14,20]. Usually states model navigation nodes, transitions model links between the navigation nodes, and events model user input or system events.

For example, the state machine in Fig. 1 models a simple online book store. Very simply spoken, the user of this book store can browse over the books (state Browse), select a book (event book) and view either its summary (BookSummary) or detail information (BookDetail), and, after successfully logging in (Login), buy the book (Buy).

However, modularization in state machine models is generally difficult, see [23]. In particular, even this simple state machine containing only seven states shows some modularity deficiencies. This is also why Fig. 1 is not very easy to comprehend. In particular, the following features complicate the state machine and obscure (at least partially) the behavior of the web application:

1. In this application, the user can go back to the home of the application from every other site. This is modeled by a transition leaving every other state to Home. This is a violation of the *Don’t Repeat Yourself* (DRY) principle.

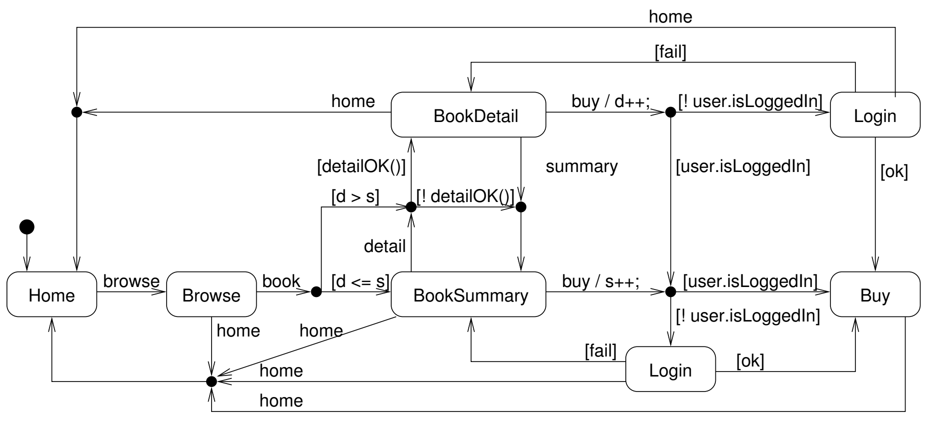


Fig. 1. Example: Modeling a book store with a UML state machine

2. In order for the user to buy a book, he is required to be logged in. Since there are two ways of entering Buy (from BookDetail and BookSummary), Login is also modeled twice. Again, the DRY principle is violated.
3. When the user finishes browsing and selects a book (event book), it is difficult to see which view, BookDetail or BookSummary is shown. In fact, two features are modeled in a tightly entangled way:
   (a) The system checks whether the summary or the detail view is more "commercially successful" and shows the user this view. That is, it stores in the two variables d and s (updated on the transitions leaving BookDetail and BookSummary) the frequencies of the user proceeding to buy the book from these views, and shows him the "right" view when the user has selected a book.
   (b) Meanwhile, the systems also checks if it is technically appropriate to show the detail view. Reasons for this view being inappropriate could be that the client device, due to factors like processing power, band width, or size of the display, does not support the display of the detail information. The detail of this check is modeled in a rather abstract function call detailsOK(). Only then this function returns true, the detail view is shown, otherwise the summary is shown instead.
4. The relation of the above two features is not easy to comprehend. Only after careful study of Fig. 1 is it clear that currently an AND relation is implemented, that is, both of the conditions must be satisfied for the book detail to be shown. Changing to OR or any other combination (e.g. the detail view should be shown as soon as it is more successful, no matter if the client is adequate or not) would be an error-prone task.

Such modularity problems of UML state machines can be addressed by HiLA. In the following, we first give a brief overview of HiLA and then show how it can be used in modeling web applications to improve the model modularity.

3 HiLA in a Nutshell

High-Level Aspects (HiLA) [23] is an aspect-oriented extension of UML state machines. It provides a new language construct to separately model parts of the system behavior, and thus enhances the modularity of the models.

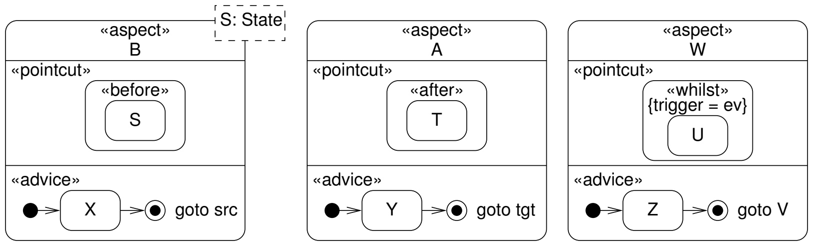
This construct is called *aspect*. An aspect is applied to a UML state machine, which is called the *base machine*, and defines some additional or alternative behavior of the base machine at some points of time during the base machine's execution. The behavior is defined in the *advice* of the aspect; the points of time to execute the advice are defined in the *pointcut*. The advice also has the form of a state machine, except that the final states may carry a label, indicating which state should be activated when the advice is finished and the the execution of the base machine should be resumed. This state is referred to as the *resumption state*. The pointcut is a specification of the points of time when certain states of the base machine are just about to become active or have just been left, or the time spans during which certain states are active. Actually, the first two kinds of pointcuts can also be regarded as those points of time when some transition is fired: a state S is just about to become active whenever any transition leading to S is fired, and it has just been left whenever a transition leaving it is fired.[2]

Overall, an aspect is a graphical model element stating that at the points of time specified by the pointcut the behavior defined by the advice should be executed, and that thereafter the base machine should resume execution by activating the state given by the label of the advice's final state. Intuitively, it can also been understood as a statement that certain transitions should be "interrupted" (what we call *advised*) by the advice.

HiLA also allows the definition of *history properties*. A history property contains a pattern, and yields the number of matches of this pattern in the execution history of the base machine. History properties can considerably reduce the complexity of the modeling of history-based features. Since HiLA is an extension of the UML, UML templates as defined in [15] can be applied to reuse HiLA aspects even more frequently.

Some examples of HiLA aspects are given in Fig. 2. Aspect B in Fig. 2(a) states that whenever state S is just about to become active («before») (that is, the aspect advises every transition leading to S), an additional state X should be activated, and then, when the final state of the advice is activated, the base machine should resume execution at the source (label goto src) of the advised transition (which means that S will *en effet* never be active). Note this aspect is defined as a template. Instantiating S with different states will specify a multitude of points of time and advise a multitude of transitions. Aspect A in Fig. 2(b) advises every transition leaving A («after»), activates state Y, and then returns to the original target (label goto tgt) of the advised transition. This aspect therefore defines an additional navigation node Y after the user has left T. Aspect W shown in Fig. 2(c) states that whenever the state U is active, and the current event is ev,

[2] Since UML state machines may actually contain concurrent regions, and there may be multiple active states at run time, pointcuts and labels actually are defined in terms of sets of states, see [23]. However, we currently do not have an example in which concurrent constructs of state machines are necessary for modeling web applications, and consider only the simple case of single-region state machines in this paper.



(a) ≪before≫ aspect (b) ≪after≫ aspect (c) ≪whilst≫ aspect

Fig. 2. HiLA Examples

the state Z should be activated, and, after that, the base machine should go to state V to resume execution.

The weaving algorithms of HiLA are described in [23]. The algorithms are prototypically implemented in HiLA/Hugo, an extension of the UML model translator and model checker Hugo/RT [12]. HiLA was applied to several case studies, including a larger-scale crisis management system, see [9].

4 Modeling Web Applications with HiLA

HiLA helps to achieve a better separation of concerns in modeling web applications as follows: first the modeler starts with a very simple state machine (which we call the *base machine*) to model the basic navigation structure. Typical hard-to-modularize features of web applications, such as *landmarks*, access control and adaption, are then modeled separately in aspects. This way, the basic navigation structure, as well as the other features, are kept simple and easy to read, hence the model is less error-prone. Potential interactions between the aspects are then resolved in a simple, declarative way.

4.1 Basic Navigation Structure

To keep the application model easy to understand and maintain, the base machine should be as simple as possible. Ideally, all information needed to determine the next transition to fire should be locally available in the source state, and there should be as little redundancy of model elements as possible. In the context of web applications this implies that the base machine should not model features like landmarks, access control, and adaptation rules. The basic navigation structure of our book store example is given in Fig. 3. The aforementioned out-sourcing of the more elaborate features makes it possible to start with a textbook state machine, i.e. one that is simple and intuitive.

4.2 Landmarks

"Landmarks" are navigation nodes that are supposed to be (directly) reachable from every other node. In order to avoid the violation of the DRY principle, some Web Engineering approaches, such as [8,16], define a keyword landmark to model landmarks.

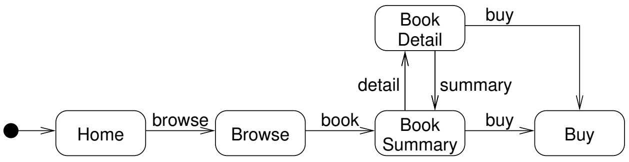


Fig. 3. Book store: basic navigation structure

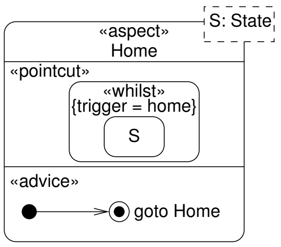


Fig. 4. Book store: aspect template defining a landmark

Unfortunately, this solution is not applicable to state-machine-based approaches, for it is not clear which event should fire the transition to the landmark. Moreover, this keyword can only be used to model landmarks that should be directly reachable from *every* other state and does not support modeling a navigation link for an arbitrary subset of the navigation nodes. In plain UML, composite states, which consist of one or more regions, are used to model common reaction of different states to the same event. Unfortunately, in the context of Web Engineering, composite states only provide partial help, but do not work if there exist landmarks which are supposed to be directly reachable from different sets of states upon different events, since a state can belong to only one region.

Using HILA, we only need a ≪whilst≫ aspect to overcome these problems, see Fig. 4: whenever state S is active (stereotype ≪whilst≫), and the current event is home (tagged value trigger = home), the advice should be executed. Since the advice does not define any behavior (the transition leaving the initial vertex leads to the final state directly), but only tells the base machine to go to state Home (label goto Home), the aspect actually state that the state machine should go to Home from state S upon event home. This aspect is defined as a UML template, instantiating the formal parameter S with different states thus models a direct navigation link from each of the states to Home.

4.3 Access Control

Access control is also hard to modularize in web application since the same logic often has to be implemented on a multitude of navigation links and this often means violation of DRY, see also [25].

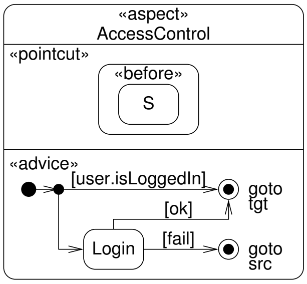


Fig. 5. Book store: aspect template for access control

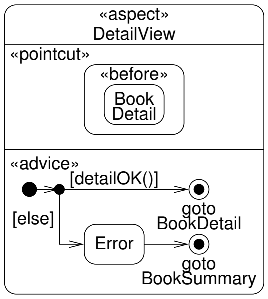
Using HiLA, a simple ≪before≫ aspect often suffices to efficiently model access control, see Fig. 5: every time just before (≪before≫) state S gets active, it is checked (in the advice) if the user is currently logged in (user.isLoggedIn). If this is the case, the final state labeled goto tgt is activated, upon which the advice is finished, and the base machine resumes execution by going to the target of the advised transition. On the other branch, if the user is not logged in, a login site is shown (Login). Again, the base machine resumes at the target of the advised transition after a successful log in (ok), otherwise the user cannot goto the state, and is "pulled" back to the source of the advised transition (label goto src).

4.4 Adaptation

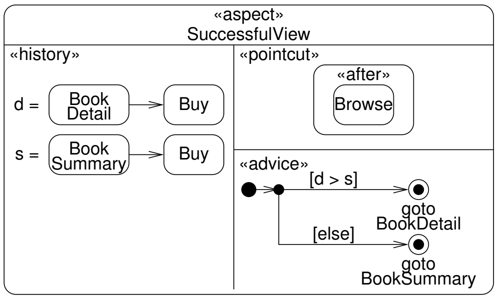
An adaptive web application is one that changes its behavior automatically to meet the preferences of the user. Adaptation is a cross-cutting concern that may easily be intertwining with other concerns of the application [1]. We show how HiLA can help to model adaptation of web applications in a highly modular fashion.

Checking whether showing details is appropriate on the client (rule 3b on page 213) before actually moving to BookDetail is pretty simple. Again, what we need is no more than a ≪before≫ aspect, see Fig. 6(a): whenever the state BookDetail is about to become active, function detailOK() is called. We do not specify this function in more detail in this paper, but simply assume that it returns true iff showing details of the current book on the client is appropriate w.r.t. the predefined conditions like network bandwidth and processing power, etc. If the function returns true, the final state labeled goto BookDetail is activated, and the base machine resumes execution at the state BookDetail, otherwise, an error message is shown (Error), and the base machine resumes execution at the state BookSummary (label goto BookSummary).

The other adaptation rule, showing the more successful view (rule 3a on page 213), defines a system behavior that is dependent on the navigation history of the user. We thus define two history properties, d and s in (the ≪history≫ compartment of) the aspect SuccessfulView (Fig. 6(b)): the pattern of d contains a transition from BookDetail to Buy. The value of d is the number of matches of this pattern in the execution history of the base machine, that is, the number of times of this transition was fired. Therefore, d is a counter of how often the user proceeded from the detail view (BookDetail) to buying the book. Similarly, the history property s counts how often the user proceeded from



(a) Checking if showing details is fine with the client



(b) Finding the more successful view

Fig. 6. Book store: two aspects for two adaptation rules

the summary view (BookSummary) to buying the book. If $\mathsf{d} > \mathsf{s}$, i.e. it is more likely for the user to buy a book when he is viewing the details of the book, then the better view is the detail view; otherwise, if it is more likely for the user to buy a book when he is viewing the summary of the book, then the summary view should be shown after the user has selected a book from Browse.

5 Feature Combination

Modern web applications usually come with an array of different features. Although it is desirable to model these features in separation, more often than not they are supposed to work together, as a whole. For instance, the two adaptation rules in our simple book store are modeled separately, but since both of them restrict the navigation to the detail view of the book, their interference has to be carefully designed.

More concretely, each of the two rules defines a constraint that has to be satisfied for the system to show the detail view of the book, but what is the relation between these two constraints? Do they both have to be satisfied? Or only one of them? For (the designer of) a modeling language, the most important question to answer is probably how easy (or hard) it is to design such a relation or to switch to another.

The UML model Fig. 1 actually implements an AND of the two rules. Only when both of them are satisfied, the system will show the detail information of the book. Comprehension of this logic requires careful study of the guards of a whole array of transitions, switching to another combination requires careful modification of the guards.

In HiLA, we therefore provides means both to detect pairs of aspects that can be interfering, and declarative ways to define their interaction.

5.1 Interference Detection

Checking whether any two aspects are possibly interfering requires checking if the labels of the final states, which finish the execution of the advices, are conflicting. Since in general an advice may contain more than one final states and not even the determination of the one that actually finishes the advice is decidable, we restrict ourselves to a conservative analysis, with the intention of finding all possible pairs of conflicting aspects, which are then subject to further investigations by more powerful instruments like model checking, or by human experts.

While the general analysis of HiLA aspects is pretty complex (see [23]), in our bookstore application, it suffices to apply a simple analysis rule: if the aspect of an aspect A1 contains a final state labeled goto G1, and another aspect A2 has a pointcut of the form «before» G1, then A1 and A2 may be interfering. Note that the two adaptation rules of the book store satisfy this condition: the advice in Fig. 6(b) contains a final state with the label goto BookDetail, while pointcut of the aspect in Fig. 6(a) has the form of «before» BookDetail. Therefore, these two aspects may be conflicting. Further investigations, for example by a human expert, show that the constraints of the conflicting labels, i.e. detailOK() in Fig. 6(a) and $d > s$ in Fig. 6(b) can actually be simultaneously satisfied, and therefore the two aspects are actually conflicting.

5.2 Declarative Feature Combination

The definition of feature combination is declarative in HiLA. After finding all potentially interfering aspects, we simply define a resumption state for each combination of the aspects. Despite an exponential complexity in theory, this procedure is often sufficiently practicable, since the sets of interfering aspects are usually small enough.

Table 1. Book store: feature combination

`DateilView`	`SuccessfulView`	Combined
goto BookDetail	goto BookDetail	goto BookDetail
goto BookDetail	goto BookSummary	goto BookSummary
goto BookSummary	goto BookDetail	goto BookSummary
goto BookSummary	goto BookSummary	goto BookSummary

In our book store, the only interfering aspects are Fig. 6(a) and Fig. 6(b). Two different labels, hence two different resumption states are defined: BookDetail and BookSummary. Table 1 shows a possible combination of these two aspects, where the column `Combined` contains the value that should actually be used. It is easy to see that only when both of the aspects set BookDetail to be the resumption state, the detail view is shown. This is the same logic as the UML solution given in Fig. 1. In contrast to the UML state machine, switching to another combination is now a simple task.

6 Related Work

Aspect-orientation has been recognized by Web Engineering researchers as helpful for improving the modularity of software design models. In [1] aspect-oriented language constructs are defined for separate modeling of adaptation. Compared with this approach, which contains only four kinds of web-specific aspects, the general-purpose language of HiLA is much more expressive.

The approaches [4,17] also propose to use aspect-oriented techniques to model adaptation modularly. Compared to HiLA, interference detection and declarative definition of aspect (feature) combination is not supported. In fact, even the more general topic, the interference between different parts of web design models, is surprisingly little investigated by the MDWE community, the only work that we are aware of being [21], which is not state-machine-based and not aspect-oriented, and our previous paper [22], which considered only navigation modeling, whereas the current paper also covers landmark and access control modeling.

Compared with other approaches of aspect-oriented state machines, the distinguishing feature of HiLA is that it is high-level. That is, HiLA aspects are defined *semantically*, based on the dynamic run time information of the base machine, whereas the other approaches, such as [5,19,26], are defined *syntactically*, based only on the static structure of the base machine. Due to this difference, HiLA aspects are simpler and easier to comprehend, and detection of interference is also easier. For a more thorough discussion, see [23].

In the MDWE context, HiLA may be applied on top of state-machine-based approaches like [2,6,14,20].

7 Conclusions and Future Work

We showed in this paper some modularity problems exhibited by UML state machines when they are used in the context of MDWE, and we showed how to use HiLA to mitigate them. In particular, using HiLA can considerably enhance the modularity and thus reduce the complexity of state machines when modeling web applications that involve landmarks, access control, and adaptation. One of the highlights of our approach is the automated detection of potential interference between the aspects, and the simple, declarative definition of feature combination.

Future work includes code generation out of HiLA aspects, and extending HiLA to model other concerns of web applications. In particular, the aspect-oriented approach of modeling rich user interface defined in [24] should be integrated in HiLA.

References

1. Baumeister, H., Knapp, A., Koch, N., Zhang, G.: Modelling Adaptivity with Aspects. In: Lowe, D., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 406–416. Springer, Heidelberg (2005)
2. Busch, M.: Integration of Security Aspects in Web Engineering. Diplomarbeit, Ludwig-Maximilians-Universität München (2011)
3. Cachero, C., Gómez, J., Pastor, Ó.: Object-Oriented Conceptual Modeling of Web Application Interfaces: the OO-$\mathcal{H}$ Method Abstract Presentation Model. In: Bauknecht, K., Madria, S.K., Pernul, G. (eds.) EC-Web 2000. LNCS, vol. 1875, pp. 206–215. Springer, Heidelberg (2000)
4. Casteleyn, S., Van Woensel, W., van der Sluijs, K., Houben, G.-J.: Aspect-Oriented Adaptation Specification in Web Information Systems: A Semantics-Based Approach. The New Review of Hypermedia and Multimedia (NRHM) 15(1), 39–71 (2009)

5. Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design. Addison-Wesley (2005)
6. Dolog, P.: Engineering Adaptive Web Applications. PhD thesis, Universität Hannover (2006)
7. Drusinsky, D.: Modeling and Verification Using UML Statecharts. Elsevier (2006)
8. Hennicker, R., Koch, N.: A UML-Based Methodology for Hypermedia Design. In: Evans, A., Kent, S., Selic, B. (eds.) UML 2000. LNCS, vol. 1939, pp. 410–424. Springer, Heidelberg (2000)
9. Hölzl, M., Knapp, A., Zhang, G.: Modeling the Car Crash Crisis Management System with HiLA. Trans. Aspect-Oriented Software Development (TAOSD) 7, 234–271 (2010)
10. Houben, G.-J., Frasincar, F., Barna, P., Vdovjak, R.: Modeling User Input and Hypermedia Dynamics in Hera. In: Koch, N., Fraternali, P., Wirsing, M. (eds.) ICWE 2004. LNCS, vol. 3140, pp. 60–73. Springer, Heidelberg (2004)
11. Int. Wsh. Aspect-Oriented Modeling (April 17, 2011), `http://dawis2.icb.uni-due.de/aom/home`
12. Knapp, A., Merz, S., Rauh, C.: Model Checking - Timed UML State Machines and Collaborations. In: Damm, W., Olderog, E.-R. (eds.) FTRTFT 2002. LNCS, vol. 2469, pp. 395–414. Springer, Heidelberg (2002)
13. Knapp, A., Zhang, G.: Model Transformations for Integrating and Validating Web Application Models. In: Mayr, H.C., Breu, R. (eds.) Proc. Modellierung (MOD 2006). Lect. Notes Informatics, vol. P-82, pp. 115–128. Gesellschaft für Informatik (2006)
14. Meliá, S., Gómez, J., Pérez, S., Díaz, O.: A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In: Schwabe, D., Curbera, F., Dantzig, P. (eds.) Proc. 8th Int. Conf. Web Engineering (ICWE 2008), pp. 13–23. IEEE (2008)
15. Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4. Specification, OMG (2010), `http://www.omg.org/spec/UML/2.4/Superstructure`
16. Rossi, G., Schwabe, D., Lyardet, F.: Web Application Models Are More Than Conceptual Models. In: Kouloumdjian, J., Roddick, J., Chen, P.P., Embley, D.W., Liddle, S.W. (eds.) ER Workshops 1999. LNCS, vol. 1727, pp. 239–253. Springer, Heidelberg (1999)
17. Schauerhuber, A.: AspectUWA: Applying Aspect-Orientation to the Model-Driven Development of Ubiquitous Web Applications. PhD thesis, Technische Universität Wien (2007)
18. De Troyer, O., Leune, C.J.: WSDM: A User Centered Design Method for Web Sites. Computer Networks 30(1-7), 85–94 (1998)
19. Whittle, J., Moreira, A., Araújo, J., Jayaraman, P. K., Elkhodary, A.M., Rabbi, R.: An Expressive Aspect Composition Language for UML State Diagrams. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 514–528. Springer, Heidelberg (2007)
20. Winckler, M., Palanque, P.A.: StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 61–76. Springer, Heidelberg (2003)
21. Wu, H.: A Reference Architecture for Adaptive Hypermedia Applications. PhD thesis, Technische Universiteit Eindhoven (2002)
22. Zhang, G.: Aspect-Oriented Modeling of Adaptive Web Applications with HiLA. In: Kotsis, G., Taniar, D., Pardede, E., Khalil, I. (eds.) Proc. 7th Int. Conf. Advances in Mobile Computing & Multimedia (MoMM 2009), pp. 331–335. ACM (2009)
23. Zhang, G.: Aspect-Oriented State Machines. PhD thesis, Ludwig-Maximilians-Universität München (2010)

24. Zhang, G.: Aspect-Oriented UI Modeling with State Machines. In: Van den Bergh, J., Sauer, S., Breiner, K., Hußmann, H., Meixner, G., Pleuss, A. (eds.) Proc. 5th Int. Wsh. Model-Driven Development of Advanced User Interfaces (MDDAUI 2010), pp. 45–48 (2010)
25. Zhang, G., Baumeister, H., Koch, N., Knapp, A.: Aspect-Oriented Modeling of Access Control in Web Applications. In: 6th Int. Wsh. Aspect Oriented Modeling (AOM 2005), Chicago (2005)
26. Zhang, J., Cottenier, T., van den Berg, A., Gray, J.: Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver. Journal of Object Technology 6(7), 89–108 (2007)