Aspect-Oriented UI Modeling with State Machines

Gefei Zhang*
Ludwig-Maximilians-Universität München
gefei.zhang@pst.ifi.lmu.de

ABSTRACT
Separated modeling of User Interface (UI) widgets is a very natural way to tackle the complexity of UI models. Due to interactions between widgets, however, this is not always an easy task. We propose an aspect-oriented approach to widget-oriented UI modeling: each widget's behavior is modeled separately in a UML state machine; synchronization of the state machines is modeled in aspects and is woven into the widget models automatically. The weaving process is transparent to the modeler. This way, we can strongly increase the degree of separation of concerns in UI modeling and reduce the complexity of UI models.

Keywords
UI modeling, UML, State machine, Aspect-oriented Modeling

1. INTRODUCTION
Modern User Interfaces (UI) are mostly interactive: the widgets are no longer supposed to be sheerly receiving input from the user or presenting the data of the system, but may also have inner lives themselves. They may have their own states, trigger events, and call other widgets to execute certain behaviors. For instance, besides acting as an input terminal for strings, a text field is more often than not supposed to be able to automatically fill in its value, or trigger database queries and fill in other widgets.

Separation of concerns in modeling such rich UI is challenging. On the one hand, it is appealing to model the widgets separately from each other, each in its own model. On the other, the synchronization of the widgets' behaviors obviously cross-cuts the widgets. An example is the requirement that only one of the widgets in a window be focused at a given time. Modeling such requirements often torpedos the natural, widget-based separation of concerns. As a result, the complexity of UI models may increase rapidly as soon as the UI gets non-trivial.

We propose an aspect-oriented approach to rich-UI modeling. Our approach enables separation of widget modeling. Each widget is modeled in its own UML state machine [11], separated from other widgets. We call the state machines *widget machines*. Necessary synchronization between widgets, if any, is left out of the widget machines, which keeps them rather simple. The synchronization is then modeled using aspect-oriented techniques: we define *aspects*, separately from the widget machines, to define constraints on or additional behaviors of the widgets. The overall behavior of the UI is then obtained by the composition of the widget machines and the aspects. We refer to the composition process as *weaving*.

In our approach, the complexity of widget synchronization is hidden behind weaving, which is transparent to the modeler. This fact and the increased separation of concerns make our approach easy to use and reduce the complexity of rich-UI models considerably.

The remainder of this paper is organized as follows: in the following Sect. 2 we present our modeling approach, including separate modeling of the widgets and aspect-oriented modeling of their synchronization; in Sect. 3 a brief discussion is given on how the aspects are woven to the state machines. Related work is discussed in Sect. 4, some concluding remarks, as well as an outline of our future research, are given in Sect. 5.

2. MODELING APPROACH
Our modeling approach is very simple. It contains three steps:

1. Construct a top-level state machine to model the basic control flow of the application, and use submachine states as place holders for widgets.
2. Model the behaviors of the widgets and complete the submachines, without considering inter-widget synchronization.
3. Define necessary aspects to synchronize the widgets.

We demonstrate this approach by means of a simple address book application. The application should provide two windows: the first containing a list of the names of all contacts and a button to show the second view, the second containing text fields for inputting data of a new contact. The second window is supposed to have a rich UI.

2.1 Top-level state machine
The first step is to model the top-level widgets, i.e., the two windows, see Fig. 1. When the application is started, the list of all contacts (`ContactList`, details ignored in this paper) is presented. The user can select to add a new contact (`newContact`), and then enter the contact details in `NewContact`. This window is supposed to have nine widgets: four input fields, four labels of the input fields, and an `OK` button to finish the input.

*Partially supported by the DFG Project MAEWA (WI 841/7–2)

Pre-proceedings of the 5th International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI 2010): Bridging between User Experience and UI Engineering, organized at the 28th ACM Conference on Human Factors in Computing Systems (CHI 2010), Atlanta, Georgia, USA, April 10, 2010.

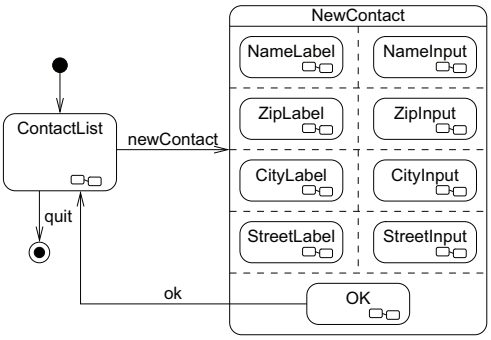

Figure 1: Top-level state machine

2.2 Widgets in Separation

The details of the widgets' behaviors are modeled separately in the sub-machines. In this step, synchronization of widgets is not considered, which simplifies the widget modeling considerably.

In our address book example, the labels have a very simple behavior: they display some predefined text, and do not react to any event. This behavior is modeled with a state machine given in Fig. 2. It contains only one state `Show`, presenting the label showing its caption.

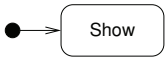

Figure 2: State machine for the label widgets

We ignore in this paper the state machine of `OK`, which is also very simple. More interesting are the state machines of the text input widgets. Figure 3 models the behavior of `NameInput`: it may be either unfocused (`NoFocus`) or focused (`WaitForInput`). If it is focused it is ready for user input (`input(t)`), and updates its text (`text = text + t`) upon each input; otherwise it does not react to any event.

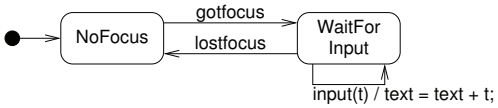

Figure 3: State machine for `NameInput`

The state machines for the other three input widgets are slightly more complex, see Fig. 4. Additionally to the behavior of `NameInput`, the other three input widgets also send the current text (`push(text)`) to whomever it concerns, and are, focused or not, ready to receive a call back (`pull(newText)` from whomever and to update the text (`text = newText`). This additional feature models the capability of automatic completion of one widget (e.g. `CityInput`) by another (e.g. `ZipInput`). Usually, getting a Zip code is only possible from a combination of a city and a street. We ignore such details here and assume they are implemented correctly in `pull` and `push`.

2.3 Synchronization by aspects

The state machines so far are simple because they do not include synchronization with each other, which is usually necessary in a rich UI. For example, the input widgets obviously are not supposed

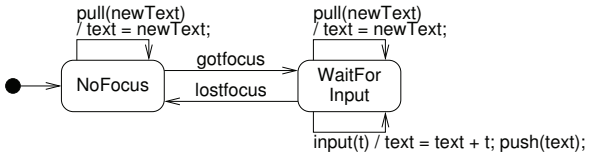

Figure 4: State machines for `ZipInput`, `CityInput` and `StreetInput`

to be in state `WaitForInput` simultaneously. Modeling such synchronization in the widget machines would break the separation of concerns, therefore we source them out and model them in aspects.

An aspect is in our approach a first-class model element. It contains a restriction to or an extension of the behavior defined in some widget machine. For instance, we model the requirement that only one widget is supposed to be in `NoFocus` by two aspects:

```
(aspect non-simultaneous
   (mutual-exclusion
      (transition NoFocus WaitForInput)))
(aspect send-others-away
   (before WaitForInput)
   (scope except-me (goto NoFocus)))
```

where the aspect `non-simultaneous` defines a restriction: only one submachine (keyword `mutual-exclusion`) is allowed to fire the transition from `NoFocus` to `WaitForInput` (keyword `transition`) at a given time. This aspect prevents the widget machines from transitioning from `NoFocus` to `WaitForInput` at the same time. The aspect `send-others-away` defines an additional behavior of the input widgets, to be executed just before (keyword `before`) state `WaitForInput` gets active: tell the others (`scope except-me`) to go to state `NoFocus` (`goto NoFocus`). These two aspects thus models the above synchronization rule concisely and separately from the widget machines.

Note that using these two aspects is not the only way of preventing the input widgets from being in `WaitForInput` simultaneously. A direct definition of mutual exclusion of states is also possible. Actually, such an aspect would be implemented as a combination of the two above aspects. We decided to use the more detailed aspects, since they are closer to the weaving (see below).

Another synchronization requirement in our sample application is that when the window `NewContact` is shown, `NameInput` should be the focused widget, i.e. in the state `WaitForInput`. We model this with the following aspect `has-focus`, which tells the submachine `NameInput` (by `scope (NameInput)`) to goto state `WaitForInput` just after `NewContact` gets active (`after NewContext.enter`).

```
(aspect has-focus
  (after NewContact.enter)
  (scope NameInput (goto WaitForInput)))
```

3. WEAVING

As simple as the aspects are, the implementation of the synchronization requires rather complex modification to the widget models.

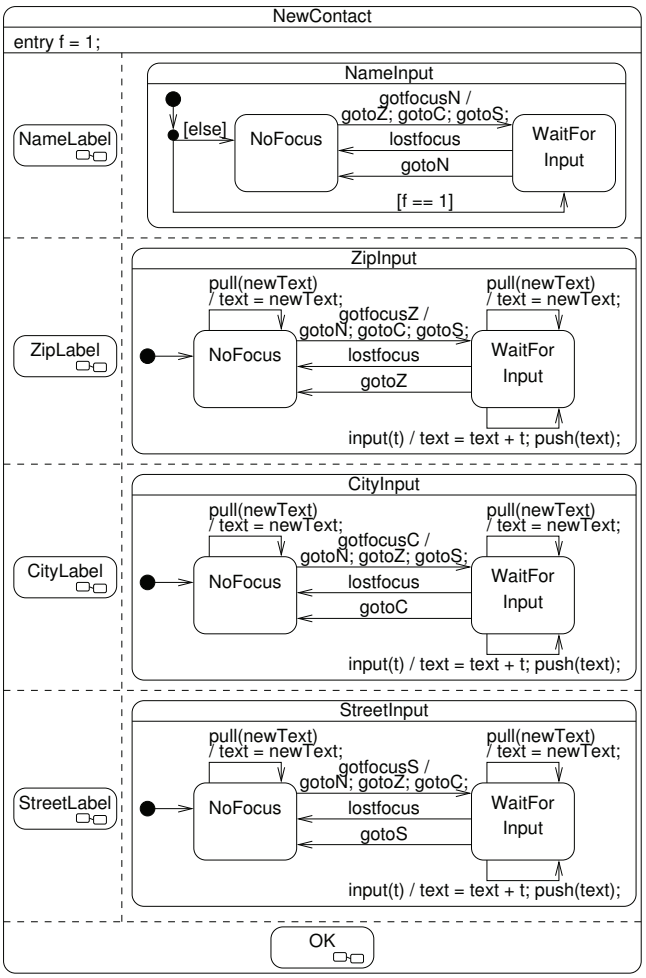

Figure 5: Partial weaving result of the sample application

This modification is taken care by an automatic weaving process, which is still ongoing work. With the automatic weaving, the aspects will be composed with the base machine "off stage", i.e., the modeler is refrained from the cumbersome details. We explain our weaving by means of the weaving result of the above aspects, see Fig. 5.

Mutual exclusion of transitions is implemented by a static renaming the events of the transitions, so that the transitions are no longer enabled at the same time. In Fig. 5, aspect `non-simultaneous` is therefore implemented by renaming the event `gotfocus` in the widget models to `gotfocusN`, `gotfocusZ`, `gotfocusI`, and `getfocusS`.

Generally, `before X` and `after X` are woven by intercepting all transitions leading to and leaving state `X`, respectively; `goto X` is woven by introducing a new transition to `X` and sending a signal to the respective state machine to fire that transition. In Fig. 5, aspect `send-others-away` is implemented as an additional transition in the widget machines from `WaitForInput` to `NoFocus`, triggered by a uniquely named event (`gotoN`, `gotoZ`, `gotoC`, `gotoS`), and an effect of the transition from `NoFocus` to `WaitForInput`, firing the "right" events.

One of the (many) exceptions to the above general rule is `after X.enter`. Obviously an interception to all transitions leaving `X` does not help in this case. Therefore, we implement `after X.enter` by introducing an entry action to `X`. In Fig. 5, aspect `has-focus` is implemented by `NewContact`'s entry action, setting `f` to `1`, and splitting the transition leaving `NameInput`'s initial vertex to active `WaitForInput` immediately if `f == 1`.

A brief glance at Fig. 5 suggests how cumbersome modeling widget synchronization may get. In comparison, modeling with our aspects is simple and straight-forward. All the complexity of actually implementing the required synchronization is hidden behind the weaving (once implemented) and invisible to the modeler.

4. RELATED WORK

Model driven development is a promising paradigm for UI development. There are several proposals of UI modeling, see [1, 5, 6, 10, 12, 17]. In particular, state machines are also used in [14, 16], where the former work describes a translation of Concurrent Task Tree (CTT) models into UML state machines, and the latter defines an extension of UML state machines to model navigation of web applications. Compared with these approaches, the distinguishing feature of our approach is its use of aspect-oriented modeling (AOM) to model synchronization of state machines. This makes the UI models of our approach easy to construct and easy to use, since the cumbersome details of interaction between widgets are hidden behind a (yet-to-implement) automatic weaving process.

AOM was also applied to reduce the complexity of design models in other application areas, such as adaptive systems [2, 13, 18] or crisis management systems [7], see in [4] for a more general overview of aspect-oriented techniques. Compared with other proposals of aspect-oriented state machines, such as [15, 21], the aspect language used in this paper is *high-level* in the sense that it is used to define modifications of behaviors on a more abstract level than (syntactical) modifications of modeling elements, see [19, 20] for a more thorough dicussion on the advantages of high-level aspect-oriented modeling.

5. CONCLUSIONS AND FUTURE WORK

We presented a widget-oriented modeling approach for interactive user interfaces. Our approach uses UML state machines, a very popular language for modeling software behaviors. By supporting aspect-oriented modeling our approach achieves a high degree of separation of concerns, and thus increases the feasibility of widget-oriented UI modeling considerably.

We plan to integrate the aspect language into HiLA[1], our general approach to aspect-oriented state machines. Using state machines as the modeling language, and in particular the definition the weaving result in the form of a state machine, makes it possible to verify the weaving result by formal methods like model checking or theorem proving. In particular, we plan to apply the UML model checker Hugo/RT [8] to verify temporal logical properties of our UI models.

6. REFERENCES

[1] Goetz Botterweck. A Model-Driven Approach to the Engineering of Multiple User Interfaces. In Thomas Kühne, editor, *Reps. and Rev. Sel. Papers Wshs and Symp. at*

[1] `http://hila.pst.ifi.lmu.de`

MoDELS'06, volume 4364 of *Lect. Notes in Comp. Sci.*, pages 106–115. Springer, 2007.

[2] Sven Casteleyn, William Van Woensel, and Geert-Jan Houben. A Semantics-based Aspect-oriented Approach to Adaptation in Web Engineering. In Simon Harper, Helen Ashman, Mark Bernstein, Alexandra I. Cristea, Hugh C. Davis, Paul De Bra, Vicki L. Hanson, and David E. Millard, editors, *Proc. 18th ACM Conf. Hypertext and Hypermedia (HYPERTEXT'07)*, pages 189–198. ACM, 2007.

[3] Gregor Engels, Bill Opdyke, Douglas C. Schmidt, and Frank Weil, editors. *Proc. 10th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'07)*, volume 4735 of *Lect. Notes Comp. Sci.* Springer, 2007.

[4] Robert E. Filman, Tzilla Elrad, Siobhán Clarke, and Mehmet Aksit, editors. *Aspect-Oriented Software Development*. Addison-Wesley, 2004.

[5] Guillaume Gauffre, Emmanuel Dubois, and Rémi Bastide. Domain-Specific Methods and Tools for the Design of Advanced Interactive Techniques. In Holger Giese, editor, *Reps. and Rev. Sel. Papers Wshs and Symp. at MoDELS'07*, volume 5002 of *Lect. Notes in Comp. Sci.*, pages 65–76. Springer, 2008.

[6] Daniel Görlich and Kai Breiner. Useware Modeling for Ambient Intelligent Production Environments. In Andreas Pleuß, Jan Van den Bergh, Heinrich Hußmann, Stefan Sauer, and Daniel Görlich, editors, *Proc. Workshop Model Driven Development of Advanced User Interfaces (MDDAUI'07)*, volume 297 of *CEUR Workshop Proceedings*. CEUR, 2007.

[7] Matthias Hölzl, Alexander Knapp, and Gefei Zhang. Modeling the Car Crash Crisis Management System with HiLA. *Trans. Aspect-Oriented Software Development (TAOSD)*, 7, 2010. Accepted.

[8] 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, 2002.

[9] Gerrit Meixner, Daniel Görlich, Kai Brainer, Heinrich Hußmann, Andreas Pleuß, Stefan Sauer, and Jan Van den Bergh, editors. *Proc. 4th Wsh. Model Driven Development of Advanced User Interfaces (MDDAUI'09)*, volume 439 of *CEUR Workshop Proceedings*. CEUR, 2009.

[10] Gerrit Meixner, Marc Seissler, and Marcel Nahler. Udit—A Graphical Editor for Task Models. In Meixner et al. [9].

[11] Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.2. OMG Available Specification, OMG, 2009. `http://www.omg.org/spec/UML/2.2/Superstructure`.

[12] Andreas Pleuß, Arnd Vitzthum, and Heinrich Hußmann. Integrating Heterogeneous Tools into Model-Centric Development of Interactive Applications. In Engels et al. [3], pages 241–255.

[13] Andrea Schauerhuber. *aspectUWA: Applying Aspect-Orientation to the Model-Driven Development of Ubiquitous Web Applications*. PhD thesis, Technische Universität Wien, 2007.

[14] Jan Van den Bergh and Karin Coninx. From Task to Dialog Model in the UML. In Marco Winckler, Hilary Johnson, and Philippe A. Palanque, editors, *Proc. 6th Int. Wsh. Task Models and Diagrams for User Interface Design (TAMODIA'07)*, volume 4849 of *Lect. Notes Comp. Sci.*, pages 98–111. Springer, 2007.

[15] Jon Whittle, Ana Moreira, João Araújo, Praveen K. Jayaraman, Ahmed M. Elkhodary, and Rasheed Rabbi. An Expressive Aspect Composition Language for UML State Diagrams. In Engels et al. [3], pages 514–528.

[16] Marco Winckler and Philippe A. Palanque. StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications. In Joaquim A. Jorge, Nuno Jardim Nunes, and João Falcão e Cunha, editors, *Proc. 10th Int. Wsh. Design Specification and Verification of Interactive Systems (DSV-IS'03)*, volume 2844 of *Lect. Notes Comp. Sci.*, pages 61–76. Springer, 2003.

[17] Andreas Wolff and Peter Forbrig. Deriving User Interfaces from Task Models. In Meixner et al. [9].

[18] Gefei Zhang. Aspect-Oriented Modeling of Adaptive Web Applications with HiLA. In Gabriele Kotsis, David Taniar, Eric Pardede, and Ismail Khalil, editors, *Proc. 7th Int. Conf. Advances in Mobile Computing & Multimedia (MoMM'09)*, pages 331–335. ACM, 2009.

[19] Gefei Zhang and Matthias Hölzl. HiLA: High-Level Aspects for UML State Machines. In *14th Int. Wsh. Aspect-Oriented Modeling (AOM@MoDELS'09)*, Denver, 2009.

[20] Gefei Zhang, Matthias Hölzl, and Alexander Knapp. Enhancing UML State Machines with Aspects. In Engels et al. [3], pages 529–543.

[21] Jing Zhang, Thomas Cottenier, Aswin van den Berg, and Jeff Gray. Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver. *Journal of Object Technology*, 6(7):89–108, 2007.