Is the UML appropriate for Interaction Design?

Giorgio Brajnik
Dip. di Matematica e Informatica,
Università di Udine
brajnik@uniud.it

April 12, 2010

Abstract

In this paper we argue that while there exist several approaches to modeling user interfaces of web applications when adopting a Model Driven Development methodology, too little attention is payed to the actual abstract behavior of the interface. In the UML-IDEA approach we adopt the UML as a modeling language and start from a very abstract view of the UI specified in terms of state machines. We claim that expressivity of the language is sufficient to capture, even at the abstract level, many usability problems. At the same time, transformations of models into more concrete levels generate executable code and test cases, wireframe prototypes and representations suitable also for accessibility assessments. In this way usability and accessibility investigations could be carried out during conceptual design.

1 Some issues in modeling user interfaces

A large fraction of user interfaces (UIs) currently being designed are targetted to the Web, thanks to its ubiquity and flexibility. In the context of web engineering there are many different approaches that can be followed in order to tackle the complex activities of requirements engineering, problem analysis and system design.

A Model-Driven Development methodology (Schmidt, 2006) can be adopted, where different types of models of the software (including the user interface) are used to represent the essential properties of the software (for example, abstract models called *Platform Independent Models*, PIMs, and *Platform Specific Models*, PSMs) and appropriate model transformation rules exist to automate or semiautomate the tasks of producing executable code or of extracting useful analysis metrics (Fernandez et al., 2009). Examples are WebML, OO-H, UMLi or UWE (Ceri et al., 2002; Gómez et al., 2001; da Silva and Paton, 2003; Koch and Wirsig, 2001). Some obvious advantages of MDD approaches are that models can capture important properties of a software that facilitate analysis, models and transformation engines can produce executable code and can support test-case generation or execution, and powerful visual modeling tools exist that shorten development cycles.

In some cases models are represented using the UML (for example, using class or activity diagrams). In other cases more specialized modeling languages are used, with a better fit to the needs of UI designers that have to cope with complex requirements like multimediality, multimodality, plasticity or retargetting, in addition to being able to model the functional aspects of the UI. This is the case of UsiXML, XIML, UIML, TERESA.(Stanciulescu, 2008; Puerta and Eisenstein, 2002; Abrams and Helms, 2004; Mori et al.,

2004)

I believe that one issue in common with many of these approaches is that, when working at the most abstract level (*i.e.*, PIMs), too little emphasis is given to modeling the dynamics of the UI. For example, WebML or OO-H models, although including aspects about the dynamics of the UI (*e.g.*, navigation model), treat the dynamics of the UI as a secondary aspect, coming after the domain model (*i.e.*, data structures) is defined. In a sense, the possible behavior of the UI is defined as a consequence of the domain model: the domain model serves as a guiding tool to design the sequence of actions users will take.

In other cases, where the emphasis of the approach is mainly on modeling the UI rather than the entire application, a large part of the modeling language is devoted to describe the structure of the UI, rather than its behavior. For example, abstract presentation and dialog elements (as in XIML), or device-independent and modality-independent UI components classes and tree-based groups of components (as in UIML). The consequence is that while the designer can avoid having to think at many detailed aspects, s/he is still forced to think of the UI in structural terms: how it is structured rather than the kind of interactions it permits.

Finally, the approaches that include task models (like TERESA or UsiXML) can focus on certain aspects of interaction (for example, temporal constraints or other dependencies among subtasks, or task allocation between user and machine). However these modeling approaches tend to be poorly integrated in web engineering methodologies. The consequence is that, right now, they are not included in mainstream MDD frameworks. One objection is that designers would have to master yet another modeling language, whose elements need to be integrated with the other languages being used (like the UML).

One consequence is that usability and accessibility of resulting interfaces are investigated at later development stages, reducing effectiveness and increasing costs of these investigations.

2 UML-IDEA

Our approach, called *UML based Interaction DEsign Approach* (UML-IDEA), aims at filling the gap: within existing MDD frameworks that are based on the UML, use the UML metamodels to focus on abstract interaction design of the UI, and let model transformation rules to semi-automatically produce structural models of the UI. One important goal of models is to be able to focus the designer attention on usability and accessibility of the UI.

Figure 1 shows the logic of the approach. Starting from computational independent models (business processes, business use cases) requirements are gathered and used to generate abstract models of the application *and* of the UI. Abstract models of the UI are based on structural models (classes and objects) of the domain – the data structures manipulated by the UI – and on behavioral models (state machines and OCL constraints) of the dynamics, building upon an idea by Thimbleby (2007).

Figure 2 shows part of these models. The case study is a web application (implemented automatically using OO-H) for managing work tasks, whereby a manager can define and assign tasks to employees, and can monitor their progress. The figure shows that, at a certain stage of the interaction, the manager can activate a control called `out of date tasks` (top left); when this happens the system selects all the tasks that are out of date, and then outputs them. The user could have activated other controls, like `New folder` or the control associated to a generic `<folder>` (not known at design time). Notice that nothing is said regarding how these controls are implemented or the modality used to execute user actions and system responses.

In another model the designer defines the domain model, in terms of classes, objects and attributes like `task`, `task list folder`, `priority`, `end date`, etc.

These models refer to abstract capabilities of the

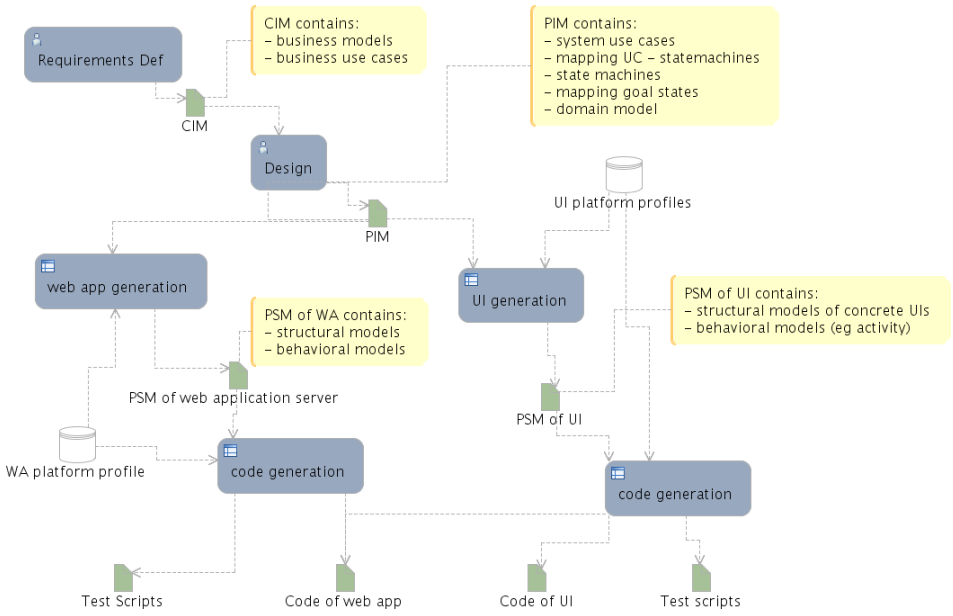

Figure 1: The MDD process flow underlying UML-IDEA.

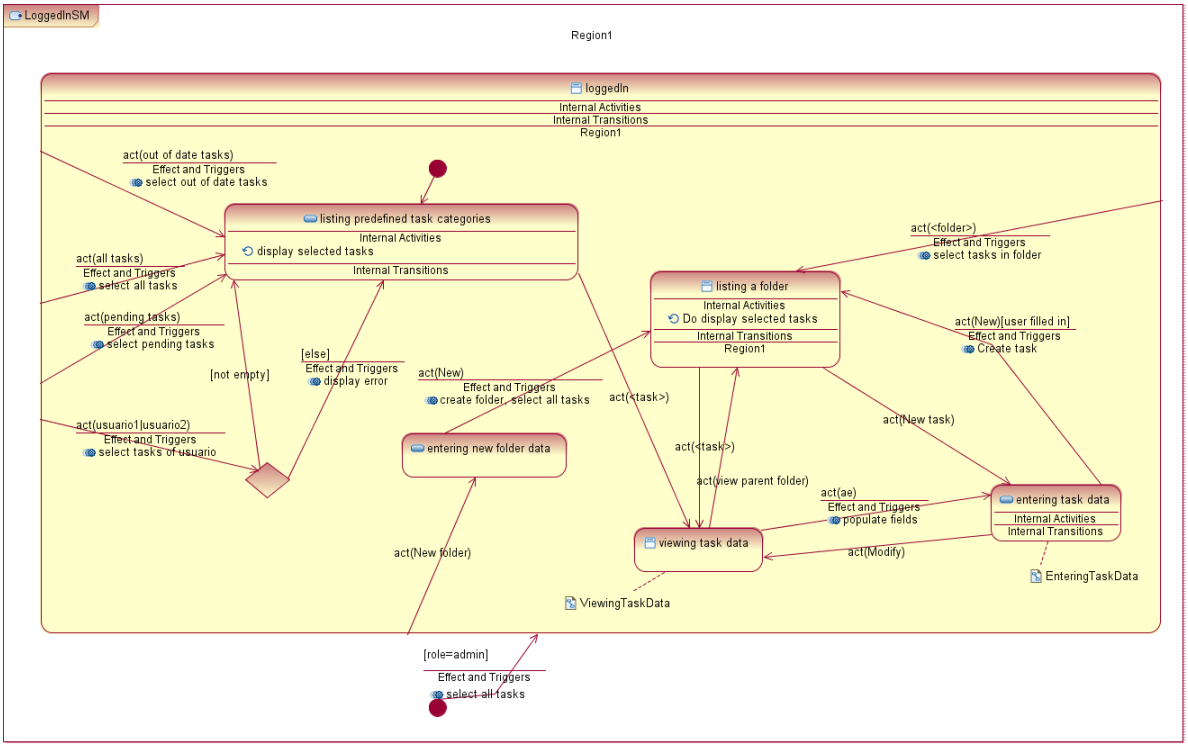

Figure 2: A state machine modeling the interactions that can occur when an administrator logs into a task management application.

UI to support an interaction, focus exclusively on the dynamics and do not commit the designer to any structural choice of the UI.

Through appropriate model transformation rules, based on capability profiles of platforms, more concrete models can be derived, eventually leading to executable code or executable test cases. In some cases, this process is relatively straightforward: for example, to build a low-fidelity wireframe prototype of the UI using static HTML and CSS, a very simple platform profile is needed, with very few templates and annotations. Figure 3 shows an HTML example corresponding to the state machine discussed above, built from the complete state machine model, the domain model and an object model specifying the domain object instances to use. The page shows the kind of conditions that are necessary in order for the UI to be in that state, the outputs displayed, and the available commands (in the case of a HTML wireframe, a list of links, grouped according to the state machine and state they are defined in).

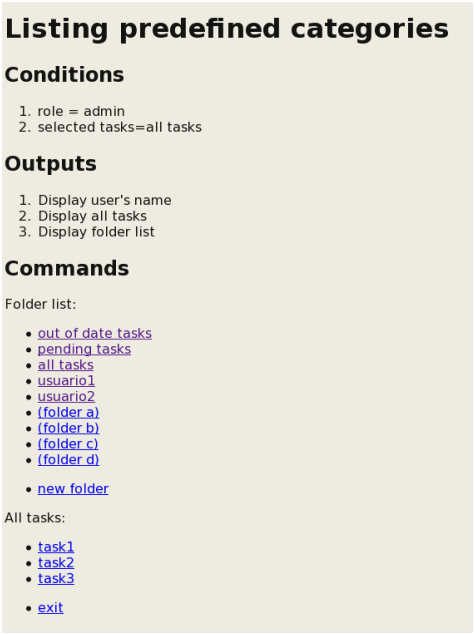

Figure 3: Screenshot of a page of a wireframe prototype built from the state machine shown in Figure 2.

3 Expected benefits

In general, the benefits of a MDD approach include: (i) support by development tools in generating executable code of the entire application and of test cases; (ii) support in retargetting the UI to different platforms; (iii) ability to use static analysis to verify models and compute useful metrics; (iv) ability to focus the developer's attention to certain point of views, and abstract away unwanted details; (v) ability to reuse design fragments, as design patterns or components.

In addition, the expected specific benefits of UML-IDEA are:

1. The adopted modeling languages are already a standard in web engineering, well supported by MDD tools, and already well known to many developers.
2. The focus of UML-IDEA is on interaction design. Models highlight the kind of information that is shown to the user and the available actions at each stage of the interaction. As a consequence the designer is required to think of the kind of decisions that the user has to make when interacting. We expect that usability could improve just because of this fact.
3. Expressivity of the models. So far we modeled various UIs of existing or realistic web applications, at various levels of abstraction (for example, down to a very detailed description of the editing commands several text fields where there is an autocompletion behavior), and existing constructs of the UML 2.0 were always sufficiently expressive.
4. The state machine diagrams can be used to reason about usability of the resulting UI. Certain types of usability problems can be easily identified from these models, or from the interaction graph obtained by flattening the state machines. An example, although trivial, should illustrate this point. If a lo-

gin error occurs in the task management application mentioned above, then a new page is shown to the user with a generic error message and to keep going the user has to click on the RETURN link, which leads to the empty login form. The usability problems we see here are: (a) the RETURN action has no utility for the user; the error message could have been shown in the same login page; (b) the login page, when shown, requires the user to explicitly set the focus to the username field, which is another user action that could be avoided; (c) the error message is opaque; it could have said "Login error. You provided username: john". All these problems can be spotted by analysing state machine models, or the induced interaction graph.

5. Ability to quickly generate, and revise, executable wireframe prototypes, to be used for debugging or for early user testing sessions.
6. Ability to statically analyse models to see whether certain properties hold on the interaction graph; for example, reachability of end user goal states, distance of goal states, existence of redundant paths, etc.
7. By using a platform where certain modalities are available, additional aspects of the interaction can be highlighted. For example, in the model shown in Figure 2 an abstract event is ACTIVATE(NEW FOLDER). Such an event in a PSM may correspond to a CLICK(NEW FOLDER) action performed with the mouse, or if a keyboard interaction occurs, to several KEYPRESS(TAB) actions to move the focus on the NEW FOLDER link, and then a final KEYPRESS(ENTER). Wireframe prototypes referring to a specific modality will highlight these differences, helping identifying accessibility problems very early in the design cycle.
8. The interaction graph can be used to identify functional test cases for the UI, generate executable code for them, and support the test management process (for example, by producing test coverage metrics).

4 Research agenda

We envision the following open research questions concerning UML-IDEA.

1. Evaluation of expressivity of the modeling language. Although well proven in software engineering, expressivity of the UML in the context of interaction design is not well studied so far. What kind of properties can be captured and what kind of conclusions can be drawn is not clear. Our experience is still too limited.
2. Feasibility of model transformations. How easily can platform profiles be defined? And writing transformation rules? How understandable, modifiable and testable these rules are? Which target languages should be used for PSM? What kind of interoperability can be envisioned at the PSM level of abstraction?
3. Static analysis of models. How far can we go with static analysis of state machines, of the induced interaction graph or of the PSMs? What kind of conclusions can be drawn that can inform the designer of possible pros and cons of a design? Which PSM metamodel is more suitable for what kind of analysis? And more specifically, for usability and accessibility?
4. Modularity of the language. Can we define design patterns? How easy is it to reuse them? Are there interdependencies between PIM and PSM? How composable are these kinds of abstract models?

Although at the moment UML-IDEA is in a preliminary stage, and there are yet no executable model transformation engines available, we are working on this (by using XMI as a representation language and writing, among others, transformation rules using XSLT), while addressing these research questions.

Acknowledgements

I'm grateful to Emilio Insfran, Silvia Abrahão and Andrea Baruzzo, my MDD and UML mentors.

References

M. Abrams and J. Helms. User interface markup language (uiml) specification working draft 3.1. Technical report, OASIS, 2004. `http://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-draft-01-20040311.pdf`.

S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, 2002.

P. Pinhero da Silva and N. Paton. User interface modeling in umli. *IEEE Software*, pages 62–69, 2003.

A. Fernandez, E. Insfran, and S. Abrahão. Integrating a usability model into model-driven web development processes. In *Web Information Systems Engineering - WISE 2009*, volume 5802/2009 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Oct 2009.

J. Gómez, C. Cachero, , and O. Pastor. Conceptual modelling of device independent web applications. *IEEE Multimedia Special Issue on Web Engineering*, pages 26–39, 2001.

N. Koch and M. Wirsig. Software engineering for adaptive hypermedia applications? In *Third Workshop on Adaptive Hypertext and Hypermedia at the 8th International Conference on User Modeling*, Germany, 2001.

G. Mori, F. Paternò, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering*, pages 507–520, 2004.

A. Puerta and J. Eisenstein. Ximl: A common representation for interaction data. In *Proc. of 6th International Conference on Intelligent User Interfaces, IUI 2002*, pages 214–215, San Francisco, USA, 2002. ACM Press.

D. Schmidt. Model-driven engineering. *Computer*, pages 25–31, February 2006.

A. Stanciulescu. *A Methodology for Developing Multimodal User Interfaces of Information Systems*. PhD thesis, Université Catholique de Louvain, Belgium, June 2008.

H. Thimbleby. *Press on: principles of interaction programming*. The MIT Press, 2007.