Models In Model-Based User Interface Design

Krzysztof Gajos
kgajos@cs.washington.edu

May 21, 2003

1 Introduction

For the sake of the clarity of my thinking, I set out to find out what kind of information usually resides in "models" in model-based user interface research. This, hopefully, will help us understand what functionality we want SUPPLE to provide and, consequently, what information we need to provide it with.

2 Goals of Model-Based UI Research

It seems that the original motivation for research on model-based user interfaces stemmed from the observation that a large portion (up to a half) of any serious program is devoted to supporting the UI. Furthermore, the UI code is brittle, difficult to maintain and, as a result, the UI is difficult to change, prototype, etc. More recently, in the advent of ubiquitous computing, the heterogeneity of presentation devices offered extra incentive for the research. Still, it is important to note that the model-based UI community seems to be concerned not only with flexible and platform-independent specification of the displayed widgets, but also with automating those aspects of the interface that have to do with the task flow, interaction with the underlying application, and so on.

The typical approach is to specify the model of the interface and then compile it into an efficient language (e.g. C, Java) and then compile the resulting code together with the rest of the application for fast execution.

Since the field is concerned with ease of development, testing and deployment, merely creating complex specification languages was not sufficient to declare success – significant portion of the effort is going into developing appropriate user interface development environments (e.g. [6]).

Another thing worth noting, is that, traditionally, presentation models are static once created and compiled. Only recently did people started noticing that certain aspects of the presentation may need to be delayed until execution time to accomodate the growing diversity of interaction devices. The ideas of Eisenstein, at al. [1] are particularily close to ours. I will summarize some of them in Section 4.

3 Classes Of Models

Model-based UI community seems to distinguish among the following major classes of models:

- **Application Model** defines the capabilities of the application. In case of [7] this means describing various classes (in the OO sense) in terms of their attributes, exceptions that methods may throw, methods together with their preconditions and, finally, a list of events published by the class.
- **Task And Dialogue Models** – although people seem to consider them separate, both [7] and [1] collapse the two into a single structure. The talk/dialogue model describes the tasks that user can perform with the system. The tasks include presenting information to the user, obtaining information from the user, invoking application functionality, etc.
- **Presentation Model** describes how the information is to be presented to the user. It seems that it describes the interface in terms of such low level elements as buttons, menus, etc.

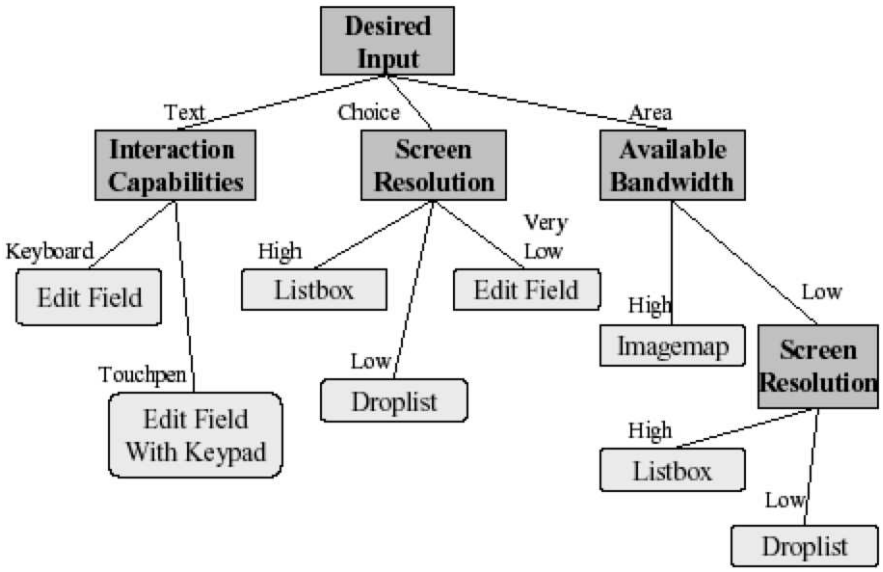

Figure 1: A sample decision tree for creating a presentation model at run time; the tree accomodates a number of constraints such as screen resolution, interaction capabilities and available bandwidth. Copied from [1]

- **Platform Model** contains information specific to the target device: i.e. its resolution, color depth, input capabilities, and so on. This level of modeling has emerged most recently in response to the need for making part of the presentation modeling dynamic.

Others expand this list even further to include the models of the user, the environment and the data [8].

The different models may be specified in different languages (e.g. application model may be encoded as an interface specification in an extension to a traditional programming language, task model may be specified as objects and methods). It seems that at least a few authors have identified method composition as a hard research problem [5]. Schreiber seems to have solved the problem by creating a single modeling system that encompases all aspects of the model in a single representation [4]. I am still reading this paper.

4 The Eisenstein, et al. Paper

This paper [1], titled *Adapting to Mobile Contexts with User-Interface Modeling*, was presented at a workshop on mobile systems and applications (shortly afterwards a somewhat less informative LPU [2] was presented at IUI'01). It opens with a description of what model-based UI generation is. It eventually points out that the presentation model is usually fixed at compile time and thus cannot be changed dynamically later. This might have been acceptable in the era of desktops but given the wide variety of mobile devices currently available, this needs to change. He shows that the presentation model could be based on *Abstract Interaction Objects* (AIOs) that can be subclassed to form instantiations for particular devices. Interestingly, these subclasses, according to the paper, should still reside in the presentation model of the application. What changes is that the right instance of an AIO is chosen at run time. He quotes many of the same constraints for choosing the right widget as those that we came up with: screen size, mouse Vs. stylus Vs. keyboard interaction. He also adds some others: available bandwidth for downloading elements and interacting with the application, and the context the device might be used in. The last point refers to the fact that the choice of modality is often dictated by the task context of the user. For example, a geologist (example from the paper) is likely to annotate a map initially on a laptop. The cellphone would most likely be used in a car for directions. Then in the field, the most convenient device would be a PDA and the most likely task – adding annotations about current location. It's great but sounds a bit far fetched to me.

One way to adapt the presentation model on the fly, is to use decision trees to incorporate various aspects of the context in which an application is running to choose the right widget instances. An example of such a decision tree is shown in Figure 1 – this seems to explain the mechanism behind the currend Pebbles rendering algorithm [3]. The authors recognize that this is not sufficient. They suggest that the ultimate solution would do something similar to what we do with putting parts of the UI into tabs. Consequently they recognize that the model should group parts of the UI into units that should be presented together – precisely what we are doing. It looks like they have never implemented such a system but in the future works section they say they plan to try a constraint-based approach and an optimization-based search.

5 Implications For Supple

I think I lack clarity as to what classes of problems SUPPLE should solve. My initial feeling was that SUPPLE should only deal with rendering the UI and all processing, including dialogue management, should be done by the application. To the extent, that if windows needed to be popped up dynamically, the application would supply corresponding abstract description and SUPPLE would render it on the fly.

5.1 Iformation Requirements of Supple – Current Thoughts

So far we have only considered the kinds of information that SUPPLE would need in order to render the interface and perform intelligent transformations of the underlying representation. Thus we consider the following classes of information:

- **Typed hierarchical description of information flowing between the interface and the application.** This is our basic model. It seems to correspond partially to the application model and partially to the abstract presentation model. Unlike other representations, however, it contains no descriptions of concrete widgets.
- **Device model** – in our case the device model includes the concrete widgets as well as the information about the screen real estate, interaction capabilities, and all other aspects relevant to rendering the interface. In our aproach, the device model is built into the rendering engine for a given platform and does not need to be specified explicitly (certainly not in a domain-independent manner). It is possible to render a UI for a PDA on a desktop machine, but the rendering has to be done by a dedicated renderer.
- **Translation Functions** – these should allow us to tranform the basic model. Currently we assume that all such tranformations would yield a simplified, less desirable (but potentially more compact) version of the interface.
- **User traces** – they are supposed to help us in creating the user model (either individual or aggregate). The user model can be used by the optimization algorithm to make choices optimized for a particular usage pattern. The traces can also help remove (or hide) unused functionality.
- **Preconditions** indicate when parts of the UI become accessible. In MASTERMIND they are part of the application model [7]. In Pebbles, they are used to decide when parts of the interface should be visible. We have argued that an opposite approach is also possible: display all parts of the interface and satisfy preconditions automatically. I have also realized, that it may sometimes make sense to express preconditions in terms of variables that are not rendered by the UI. Thus we need a fourth category of variables (in addition to settable, readable and read/settable).
- **Magic model** – I will not assign a specific name to this until we decide what should be in it. This is the magic causal model what we planned to use to infer how to simplify the interface automatically. We have argued that it may need to contain constraints or planning operators indicating how the variables interact with one another.

References

[1] Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta. Adapting to mobile contexts with user-interface modeling. In *Workshop on Mobile Computing Systems and Applications*, Monterey, CA, 2000.

[2] Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta. Applying model-based techniques to the development of uis for mobile computers. In *Proceedings of the 6th international conference on Intelligent user interfaces*, pages 69–76. ACM Press, 2001.

[3] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joe Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. Generating remote control interfaces for complex appliances. In *CHI Letters: ACM Symposium on User Interface Software and Technology, UIST'02*, Paris, France, 2002.

[4] Siegfried Schreiber. Specification and generation of user interfaces with the BOSS-system. In *EWHCI*, pages 107–120, 1994.

[5] Kurt Stirewalt and Spencer Rugaber. Automating ui generation by model composition. In *ASE*, pages 177–, 1998.

[6] Pedro Szekely, Ping Luo, and Robert Neches. Beyond Interface Builders: Model-Based Interface Tools. In *INTERCHI'93*, pages 383–390, April 1993.

[7] Pedro A. Szekely, Piyawadee Noi Sukaviriya, Pablo Castells, Jeyakumar Muthukumarasamy, and Ewald Salcher. Declarative interface models for user interface construction tools: the MASTERMIND approach. In *EHCI*, pages 120–150, 1995.

[8] Jean Vanderdonckt and Pierre Berquin. Towards a very large model-based approach for user interface development. In *UIDIS*, pages 76–85, 1999.