Integrating dialog modelling and application development

Hallvard Trætteberg
Norwegian University of Science and Technology
7491 Trondheim, Norway
+47 7359 3443
hal@idi.ntnu.no

ABSTRACT
Based on a set of characteristics for diffusion of technology, we question the current state and direction of MBUID. We have suggest a set of requirements based on this set of characteristics and present how our own work relate to this.

Keywords
Technology diffusion, model-based methods, industrial practice.

INTRODUCTION
Model-based (MB) user interface design (UID) has been a promissing approach for over a decade. In fact, MBUID has been promissing for so long, while not being adopted by the mainstream application developer, that we perhaps should question the approach. Not because we believe models are of no use, but because we have observed that the envisioned and suggested role of models and modelling do not fit current industrial development practice. And to be honest, it doesn't fit my own practice either.

A recent ACM article on adoption of the iMode wireless technology [1], presents a set of innovation characteristics, developed by Rogers [2] to explain adoption of new technology. The characteristics are explained as follows (quoting from the ACM article, not the Rogers' work):

- Relative advantage: the degree to which the innovation is perceived as being better than the practice it supersedes.
- Compatibility: the extent to which adopting the innovation is compatible with what people do
- Complexity: the degree to which an innovation is perceived as relatively difficult to understand and use
- Trialability: the degree to which an innovation may be experimented with on a limited basis before making an adoption (or rejecting) decision
- Observability: the degree to which the results of an innovation are visible to others

The characteristics are compatible with the simple observation that change (in people) happens gradually and only when the evidence is clear. In this paper we discuss these characteristics in the context of methods and tools for MBUID.

INNOVATION CHARACTERISTICS FOR MODEL-BASED TOOLS AND METHOD FOR USER INTERFACE DESIGN
For the sake of this discussion, we will assume that developers currently use UML tools for supporting design and implementation, use GUI-builders to a limited extent, utilise standard and third party component libraries/frameworks and do a lot of hand-coding for meeting "special" requirements. Prototyping is used for discussing design alternatives, and often parts of the prototype evolve into shipping code, usually due to lack of time for recoding. All this is based on our personal (and limited) contact with industry.

Relative advantage: The advantage of the model-based approach may be related to characteristics of the resulting user interfaces or the development process (or both), e.g. increased usability or decreased development time or resource consumption. State-of-the art MBUID tools should in many cases result in less development time, if we disregard the time required to learn the methods and tools. However, most tools have limited coverage of interaction styles and platform technology, and going outside the supported set is usually difficult and time-consuming, if not impossible. Another possible advantage of MB tools is that existing design models are easier to utilise, so MBUID needs to do less. However, our experience is that the software design models are not user-centered and as the starting point for UID should be used with care. In addition, few UI models are based on UML, which is the only candidate for de-facto software modelling standard.

Compatibility: The idea of using models is catching on within systems engineering, as the penetration of UML shows. Note however, that software modelling concepts are more similar to programming concepts, than UI models are to UI program code. Hence, it is difficult to argue that UI modelling is compatible with current modelling practice.

The model-based approach is usually taught as a top-down process of model refinement and transformation. This is contrary to a user-centered prototype-based approach, where design is bottom-up. We admit that few follow the guidelines of UCD, but have little reason to argue against it.

Complexity: Our experience in teaching MBUID to students and industry indicates that it is indeed perceived as difficult to understand and use.

Trialability: First, it is difficult to use the MB approach for limited parts of an application, since most MB approaches require changing most of the development process. Second, even if limited parts are handled well with a model-based approach, it is difficult to argue for the robustness and scalability of the approach.

Observability: I have never seen an application with a "developed using models" or "models execute here" sticker or ad (like "Intel inside" on PC hardware). Seriously, models have no visible positive effects on usability, e.g. increased flexiblity, tailorability or adaptability, which are areas where models are expected to give a positive effect. I have seen applications that give me reasons to believe models have been used for a positive effect (typically for supporting advanced customisation), but it has never been a selling point.

I may sound very negative, but I think that even such a simplified analysis may provide some lessons. Based on each characteristic it should be possible to generate requirements for the next generation of MBUID tools and methods.

INNOVATION CHARACTERISTICS AND REQUIREMENTS FOR TOOLS AND METHODS

Based on each characteristic, I will briefly suggest how it should affect the MB approach to UID.

Compatibility: Make models work better with prototyping techniques. Support processes where models are derived/built bottom-up from concrete designs, not just top-down. Models must complement concrete representations and their relation must be clear. Augment GUI-builders with MB functionality. Integrate concepts from UML into UI models, where possible, e.g. by adding UML stereotypes or extending the UML meta-model. Make it easier to integrate MB runtime systems into applications based on standard toolkits.

Complexity: Simplify modelling languages and notations.

Trialability: Design methods that may be used for limited parts of a design/project. Build open-source tools that make methods easier to try and that may be integrated into existing ones.

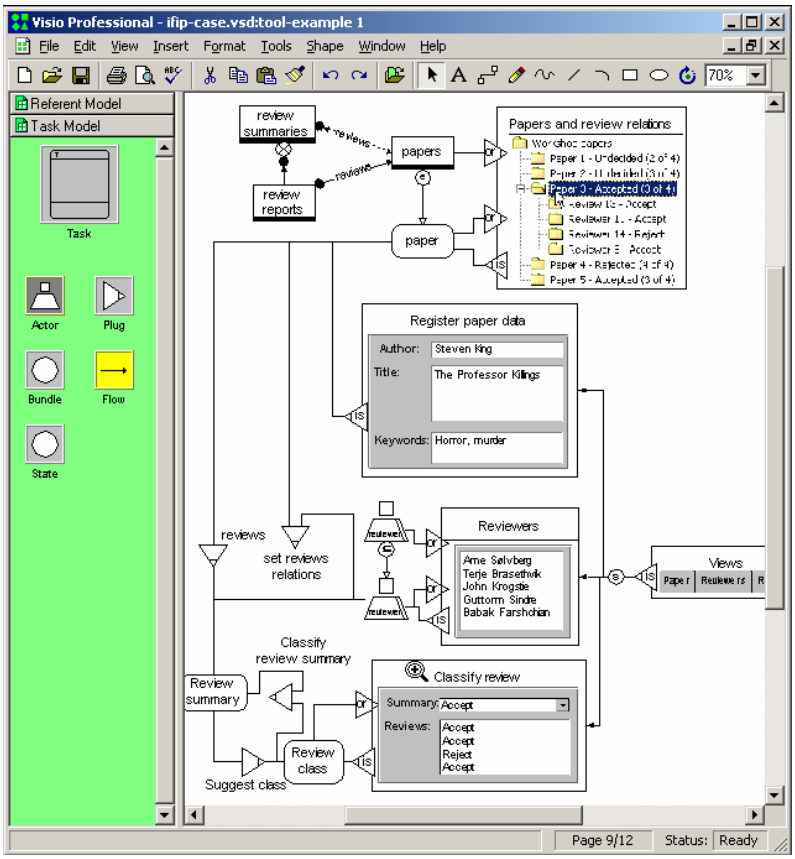
Relative advantage: Focus on the areas where most is gained, e.g. flexibility/tailorability. Make models complement existing models and design representations, and provide means for moving between them. Make models useful for smaller parts of a project.

OUR OWN APPROACH

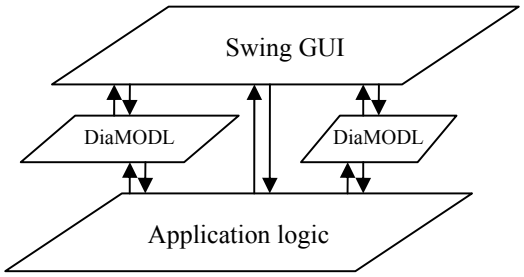
We have by no means followed all the suggestions outlined above, but most of our work on dialog modelling take the above reasonning into account.

Dialog modelling language: Our visual dialog modelling language, DiaMODL [3], is based on UML statecharts and a simplified variant of the interactor user interface component abstraction. Standard UML class and collaboration diagrams are used for domain modelling. We have modelled most (if not all) standard widgets, so the relation between AIOs and CIOs is well understood.

Modelling tool: We are working on a hybrid GUI-builder and modelling tool, based on the mock-up design shown above. The basic idea is to start with a GUI-builder and add value with DiaMODL constructs, so more of the underlying logic of the GUI may be expressed using the tool instead of by coding. The user should be free to view the design as GUI only, model only or a hybrid view as shown above. Executing should be directly supported, whatever view is used.



Using DiaMODL in applications: To make DiaMODL easier to use as part of a larger application, we have implemented the main modelling concepts in Java, and integrated it with the Swing toolkit. The model object structure is completely complementary to Swing's component structure. The former is driven by and drives the latter, does not replace or hide



it, as illustrated above. The machinery is designed so that smaller or bigger parts of the user interface may be driven by the model objects, without affecting the other parts. Hence, it is possible to gradually introduce model objects into an existing application, and try out the MB approach for only smaller parts of a new design.

XML as external format: We have designed XML languages for both Swing component structures and DiaMODL objects, to make both easier to work with. The Swing XML elements and attributes translate directly to Java objects and properties, so the Java programmer gets what she expects. E.g. the first fragment below instantiates an instance of javax.swing.JSlider and sets six of its properties.

```
<slider id="result-component"
       value="11" minimum="0" maximum="20"
       major-tick-spacing="2" paint-ticks="yes"
       paint-labels="yes"/>

<component-interactor
        idref:component="result-component"
        output-receive="Integer"
        input-send="Integer"/>
```

DiaMODL objects are built using separate XML fragments, that refer to the Swing components using XPath and identifiers. The second fragment above augments the slider with a description of its logic function, and makes it possible to present data and get input by means of the slider without "knowing" the details of how the slider is configured to handle Integer input and output.

The Java application may operate on the Swing part directly, and it is possible to use the Swing XML language for parts of the GUI that are not affected by the dialog model, as an easier and more flexible way (it may be changed without recompiling the application) for instantiating it. Alternatively, the application may operate on the more abstract model objects, e.g. provide information and react to abstract events, and hence be more robust to changes in the concrete (surface) design.

Open-source implementation: Everything we do is based on freely available open-source libraries, e.g. SAX, XOM and Jaxen for XML handling and JGraph for diagram editing, and our own code is open-source (currently residing in our department's open-source portal similar to SourceForge). This means people are free to try it out, integrate it into their own applications and modify it as needed.

Industry-friendly method: Our industrial experience shows that developers and designers are vary of being too formal. We have arranged several design workshops where we try to show how models may complement low and high fidelity prototypes. Based on this experience, we are working on integrating DiaMODL in a semi-formal approach, and have chosen Constantine's abstract prototypes [4] as a starting point. The advantage of his method, is its step by step introduction of detail and formality. Gradually adding modelling constructs when the precision provided by models is needed, seems like a natural extension of his abstract prototypes.

CONCLUSION

Based on a set of characteristics for diffusion of technology, we have questioned the current state and direction of MBUID. We have suggested a set of requirements based on this set of characteristics and presented how our own work relates to this.

REFERENCES

1. Barnes, S. J., Huff, S. L. *Rising Sun: iMode and the Wireless Internet.* Communications of the ACM, 46, 79-84, 2003.
2. Rogers, E. *Diffusion of Innovation.* Free Press, New York, 1995.
3. Trætteberg, H. *Dialog modelling with interactors and UML Statecharts - a hybrid approach.* Presented at DSVIS-2003 in Funchal, Madeira.
4. Constantine, L., Windl, H., Noble, J., Lockwood, L. *From Abstraction to Realization in User Interface Designs: Abstract Prototypes Based on Canonical Abstract Components.* Paper found at www.foruse.com.