ANALYSIS & DESIGN OF MODEL-BASED USER INTERFACES

An approach to refining specifications towards implementation

Pedro J. Molina and Hallvard Trætteberg
CARE Technologies S.A., Spain. : Norwegian University of Science and Technology, Norway.

Abstract: This paper proposes a method for user interface development where a model for analysis (Just-UI) and a model for design (DiaMODL) are conveniently combined into an integrated method. Just-UI currently supports automatically refining analysis models, through conceptual patterns to concrete user interface designs. Integrating a dialog modeling language (DiaMODL) into the method, will let the designers take part in the refinement process, hence gaining control and allowing a greater variety of designs. The method encourages the use of code generation for rapid prototyping of the UI. The ultimate objective is to provide a suitable software engineering and user interface design method with coverage to cross through requirements to final implementation of core application code and user interface.

Keywords: model based user interface development, conceptual models, design models, user interface development

1. INTRODUCTION

Model based user interface development (MB-UID) is a field mature enough to be applied into the software industry. However, still now, MB-UID methods have produced little impact into the way user interface developers create user interfaces. Such methods are difficult to introduce due to different problems: learning curve, scalability, expressiveness, lack of adequate tools, code immaturity of generators, etc.

In this context Novak’s rule [9] becomes a painful truth: people will not use specifications if the spend more time and resources creating the specification and development from such a specification than developing in
their classical approach (without formal specification). In other words, using a model-based approach has to be perceived by developers as an immediate gain (in terms of productivity, quality, resource saving, error reduction, better documentation, ease of maintenance, reduction in time to market, etc.) and not as a useless intermediate step (waste of time, outdated documentation). Therefore, we definitely think that more effort must be put into creating agile methods and tools that are seen as really useful by actual developers and facilitate the adoption of MB-UID in the software industry.

In this work, we try to integrate the advantages of using two proven models for user interface development: one of them for analysis (JUST-UI [7]) and another one for design (DiaMODL [15]), establishing a bridge from the former to the latter. Supported by code generation, the proposed method intends to be interesting not only from the academic point of view, but also useful to develop user interfaces for commercial software, thus increasing the adoption of MB-UID methods and tools in the software industry.

2. RELATED WORK

TRIDENT [16] is a reference project in the field of MB-UID tools. A Task Model (Activity Chain Graphs), a simple Domain Model (extended entity-relational model) and Dialog Model are the main artifacts employed to derive user interfaces in a semi-automatic way. Designers can participate in the process choosing design alternatives among presented by an expert system during the process of mapping from Abstract Interaction Objects (AIOs) to Concrete Interaction Objects (CIOs) (SEGUIA [1]).

MECANO, MOBI-D and MOBILE [10,13,14] are projects that have had a significant impact in the state of the art of MB-UID tools. In these tools, domain, dialog, task, user and presentation models are used on the design process.

Just-UI [7] is the model of OO-Method [11] for the analysis of business user interfaces. It is named “Presentation Model” in OO-Method. This model is based on a conceptual user interface pattern language. It is used to specify UIs from requirements in an abstract way: no design choices or platform considerations are taken. The presentation model complements the domain model resulting from object-oriented analysis of the functionality of the system. OO-Method, including this model, is supported by two commercially available tools: OlivaNova Modeler and Transformation Engines (code generators) to produce implementations for several devices [2]. The model has been used during the last four years in an industrial context and has increased the developer productivity [5].
DiaMODL [15] is a hybrid dialog modeling language based on interconnected abstract interaction objects and UML class/object diagrams and Statecharts. DiaMODL may be used for documenting the function and structure of concrete UIs, abstract specification of UI functionality and for systematic exploration of design alternatives. DiaMODL has also been used for capturing design knowledge in abstract design patterns [17].

PSA (Patterns Supported Approach) [6] is a methodological proposal to use patterns during the whole software life-cycle: from requirements to implementation. The main idea behind the approach is that patterns constitute proven solutions to frequent problems. Patterns are pills of distilled experience and constitute a common lingua franca [4] shared by the development team. PSA is not specifically tied with user interface production. On the contrary, it can be applied to general software development.

3. THE PROPOSAL

The method we propose comprises the following steps:
1. Task Analysis and Use Case specification (to organize requirements);
2. User Interface Analysis (using Just-UI);
3. Refine analysis model to abstract design model expressed in DiaMODL;
4. Abstract design to concrete UI specification; and
5. Code Generation and Implementation (producing an executable UI).

The general idea is to gradually move from problem-oriented representations to solution-oriented ones as Figure 1 suggests. Starting with the problem i.e. the user domain and tasks, the design process breaks down user tasks, moves through abstract descriptions of design to more formal, detailed and concrete descriptions of a solution. By breaking the whole process into smaller steps, we gain several advantages:
- it is easier to apply appropriate design knowledge (whether automatic or manual)
- the process becomes more transparent, since the representation at each step may be examined and manipulated by the designer/analyst
- notations and tools may be tailored to each step
Pedro J. Molina and Hallvard Trætteberg

- it is easier to manage changes, as it is easier to trace the relations between representations at different points from problem to solution.

The steps will be explained in the following subsections accompanied with a small design example to illustrate the concepts.

![Diagram](image.png)

Figure 2. Example of User Task expressed using a Use Case.

### 3.1 Task Analysis / Use Cases

A task analysis supported by use cases is a good starting point for discovering user goals in a given system. Main objectives and secondary tasks can progressively be discovered and organized in convenient use cases. These use cases can also be prioritized accordingly to the relevance of the functionality in the system. Care should however be taken when developing the use cases, to avoid a system-centric view. I.e. focus should be put on the user actor’s need, rather than on system response.

Figure 2 shows an example of high-level user task described with a use case. This example will be used to illustrate the following explanations. In the use case, the main task of a Program Chair is to Accept or Reject the papers for a given conference or workshop. As secondary tasks, selecting and browsing papers will be also required to accomplish the first class tasks. In this way, task requirements of the system are depicted: from high-level goals to more concrete user actions that will require interaction with the system through a UI.

### 3.2 User Interface Analysis with Just-UI

As described in the related work section, Just-UI [7] will be used in the stage of UI analysis. The main objective of this phase is to take requirements into account and focus only on what is needed instead of how it will be realized. Here, conceptual user interface patterns as presented in [8] helps to describe the desired UI. UI patterns represent common configurations of UI elements that support typical user tasks, such as exploration, navigation, search, selection and service invocation. In Just-UI three levels of UI patterns have been identified, and patterns are structured in a hierarchy to represent how lower-level patterns may be used to build and/or augment
higher-level ones. A chart of conceptual user interface patterns used in this approach is shown in Figure 3.

Figure 3. Just-UI pattern language.

One of the central concepts in Just-UI is the Interaction Unit concept (IU). It represents an abstraction of a window, a web page or any other component supporting communication between human and machine. Interaction Units have been categorized as patterns comprising not only presentation aspects, but also behavior and semantics based on the tasks to be accomplished in each unit. Each pattern shown in Figure 3 is such an interaction unit.

During the specification, prototypes derived by transformation engines allow to validate requirements with customers in a few and very fast cycles of requirements-gathering, specification and user validation (one-three days per cycle, two-seven iterations). Tool support and code generation for rapid prototyping is crucial to guarantee the agility of approach.

Patterns are used as primitives to build a user interface specification. To avoid mixtures of different metaphors, interaction styles and conventions, patterns are tailored for specific domains. In the case of Just-UI, the domain of is user interfaces for business application and information systems. Note
however that the approach itself is not particular to a given domain. Only the pattern language used (Just-UI) is oriented towards business applications.

Figure 4. Example of Just-UI diagram.

The pattern language constrains which patterns are meaningful to apply together, in the general case. An actual design for solving a particular use case, is specified by a structure of patterns forming a part-of hierarchy and linked by navigation links, in what is called a Just-UI diagram.

Figure 4 shows an example of specification solving the use case described in Figure 2. A population of papers (IUP_Papers) allows browsing among a list of papers. When a paper is selected, information and reviews of that paper appear in a master/detail interaction unit (IUMP_Paper_And_Reviews). This unit is composed of an instance (IUI_Paper) as master component and a population (IUP_Reviews) as a detail. Finally, two services can be launched for the paper: accept and reject, using two service interaction units (IUS_Accept and IUS_Reject respectively).

3.3 Refine analysis models to abstract design models

The patterns and Just-UI diagrams are fairly high-level and are similar to Constantine’s abstract prototypes [3] in that respect. Nevertheless, it is possible to generate code from Just-UI diagrams, if we let the generator take a lot of design decisions. This is acceptable for the purpose of quickly showing end-users a prototype, but the designer should have the option of taking part in the design process.

To give the designer this possibility, the Just-UI diagrams are converted to DiaMODL diagrams, which may be restructured, elaborated, detailed and otherwise manipulated, before later being mapped to a concrete design. The
DiaMODL diagrams make it easier to systematically explore alternatives for the design details that are underspecified in the Just-UI diagrams and decisions made by the UI generator.

![DiaMODL Diagram](image)

Figure 5. Corresponding DiaMODL diagram.

The example above may be modeled by the DiaMODL diagram shown in Figure 5. The **IUP_Papers** interactor presents a set of papers (UML multi-object) to the user and the user may select one of the papers. The selected paper flows into the **IUMP_Paper_And_Reviews** interactor, where it is presented to the user by the **IUI_Paper** interactor. In addition, a set of reviews are computed from the paper by the **paper-reviews** function, and presented to the user by the **IUP_Reviews** interactor. Finally, the same selected paper is used as input to the **IUS_Accept** and **IUS_Reject** functions, which may be triggered by the user (the triggering event is not shown in the diagram).

We see that each basic Interaction Unit in the Just-UI diagram has a corresponding DiaMODL fragment. These fragments are pre-made, i.e. for each of the basic IUs defined by the Just-UI pattern language, we have made corresponding DiaMODL fragments. Just-UI’s compositional operators are similarly mapped to DiaMODL’s interactor composition and gate interconnection operators, which are used to compose these fragments. This process is mechanical, although currently manual. We are investigating doing the composition automatically, but the result will nevertheless only be a suggestion that the design may want to edit.
3.4 Elaborating DiaMODL diagrams

A DiaMODL may be more or less explicit about design details, and the designer may want to both add detail and change the existing design. This can happen in at least two ways, as we will illustrate below.

First, the decomposition of an interactor may be incomplete (or not present) and the designer may want to add functionality in the decomposition. E.g. the GUI generated for the IUP_Papers interactor in the example above, will typically be a listbox with single selection. It may however be relevant to include a search field for limiting the size of the list, in this case the number of papers in the list. Hence, the designer will decompose IUP_Papers and add functionality corresponding to a relevant use case.

It is for instance, not uncommon that a reviewer is late with his reviews and that the program committee will try to accept/reject papers without the missing reviews. This use case may be supported by letting the user enter the reviewer’s name and limit the list of papers accordingly. A possible decomposition of IUP_Papers is shown in Figure 6.

![Figure 6. Decomposition of IUP_Papers.](image)

The search field is represented by the IU_Reviewer_name interactor. From the entered name and the set of all reviewers (the reviewers multi-object) a specific reviewer is computed by the Find_reviewer function. A second function (Papers_for_reviewer) computes the limited set of papers based on this reviewer, and the result is presented by the IUP_Papers_2 interactor. Note that this decomposition preserves the outer IUP_Papers interactor’s gate interface, and that the inner
IUP_Papers_2 corresponds to the original IUP_Papers interactor, and hence maps to a single selection listbox.

A second way of elaborating the original model, is adding sequencing details. E.g. the diagram in Figure 5 is not explicit about when the different Interaction Units (or interactors in DiaMODL terminology) are active in the user interface. One possibility is one large window with the IUP_Papers, IUI_Paper and IUP_Reviews are all present simultaneously. This is acceptable for desktop screens and supports locating a paper based on both paper info and reviewers. A medium-sized possibility is adding a transition from IUP_Papers and IUMP_Paper_And_Reviews, so the latter replaces the former when a paper is selected and a trigger is activated. This is shown left in Figure 7. It can be argued that the designer should not have to work at this level of detail, and we agree: The pattern and default DiaMODL diagram provides the (most) common default case, while the designer has the freedom to edit the result, e.g. based on user preferences and the target platform.

![Diagram](image)

Figure 7. Two variants of interactor activation logic.

### 3.5 Concrete UI specification

The DiaMODL diagram elements must be mapped to concrete UI elements, to make the design complete. For each basic IU an appropriate CIO must be found, usually selected among several candidate CIOs. In the example, the IUs presenting sets of objects (IUP_Papers and IUP_Reviews) correspond to list and table interaction objects. The chosen CIO for IUP_Papers must support single selection, while IUP_Reviews does not specify a need for selection. The chosen CIO will lead to additional design choices, like which columns to include, fonts and colors to use etc.
The next design step is to select CIOs for triggering action invocations, e.g. whether to use menu items or buttons for invoking the accept-paper and reject-paper object services. Furthermore, rules for triggering transitions between IUs must be specified. If the right variant in Figure 7 is chosen, triggers for IUMP_Paper_and_IUP_Reviews and for IUP_Reviews must be specified, e.g. return, double-click and/or a button for the former and a button for the latter.

This mapping process is fairly mechanical, in that the DiaMODL diagram may be used to drive a systematic mapping process. The mapping may be supported by tools, and in many cases diagram elements may be automatically mapped to CIOs. Design patterns may be applied to quickly get a reasonable default design, while the designer is free to edit the result, based on user preferences and target platform constraints and conventions.

3.6 Code Generation / Implementation

Just-UI is supported by a modeling tool and a set of code generators (OlivaNova Model Execution System) to produce user interfaces for different devices (such as desktop and web) directly from an analysis Just-UI specification. This approach has been useful in an industrial environment to produce code for commercial applications.

![Generated Desktop UI for the example task.](image)

An example of code generated for the case of study directly from a Just-UI model is shown in Figure 8. It contains a UI capable of reviewing papers,
revisions and accepting or rejecting the paper. The generated UI is suitable for early validation of user requirements. However, specific design aspects of the user interface code has to be tuned manually due to the lack of a design model capable of expressing different choices, alternatives to the common default values for design choices.

Therefore, the approach presented here, proposes to use DiaMODL just in the point where Just-UI jumps into the implementation using code generators. Refined specifications using DiaMODL can be changed by designers to take in account alternatives designs and selecting platform properties like font, colors, CIO selection and so on. In this way, design models will contain more detail information that the analysis one: enabling a finer tuning of the generation process to suit better the UI needed. DiaMODL is currently integrated with the Just-UI only, and not with the OlivaNova tool. A Java execution engine for the DiaMODL language exists, but it currently cannot generate code and do not support other platforms, as OlivaNova does.

4. CONCLUSIONS

The method proposed follows a gradual approach focusing on requirements, what (analysis), how (design) and implementation using a continuous and model-based approach. Each step has a clear focus, clear separation of concerns and uses as input the output of the previous steps.

A model-based approach for user interfaces has been used, refining a device independent analysis specification to a design specification and progressively reaching a device dependent specification taking in account the capabilities of the target device. This is compliant with modern engineering techniques like Model Driven Approach [10] where Platform Independent Models (PIMs) are refined to specific ones (PSMs) and finally are automatically or semi-automatically converted to source-code.

The approach presented enriches engineering methods for producing user interfaces in the way that covers analysis and design of user interfaces. With more design information the generated code will better fit the application and target platform.

REFERENCES