Automatic Generation of User Interfaces from Domain and Use Case Models

António Miguel Rosado da Cruz
Higher School of Technology and Management of the Polytechnic Institute of Viana do Castelo
antonio.cruz@acm.org

João Pascoal de Faria
Faculty of Engineering of the University of Porto
jpf@fe.up.pt

Abstract

In this paper, we envision an approach for the automatic generation of a user interface (UI) prototype from a system domain model (or core model), that captures the main domain entities and transactions, and a system use case model, which captures the intended user tasks. This prototype allows the early validation of executable core system models, and can be used as a basis for subsequent developments.

The envisioned solution uses OCL to add preciseness and semantic richness both to the domain and use case UML models. The generated UI provides some usability enhancements that are derived from the model's pre-conditions.

1. Introduction

Software development processes have been using, for decades, software models that help software engineers cope with complexity, when analyzing the problem domain or designing a software solution. Software models capture relevant parts of the problem and solution domains and are typically used as a means for reasoning about the system properties and communicating with the stakeholders.

A common software engineering practice is to build a UML system model, comprising a *domain model* and a *use case model*, supplemented by a non-functional *user interface prototype*, in the early states of the software development process [7]. The domain model captures the main system's domain classes, its attributes, relations and, in some cases, its operations, through UML class diagrams. The use case model captures the main system functionalities from the user's viewpoint through UML use case diagrams and accompanying textual descriptions. The user interface prototype is used to elicit and validate requirements with end users and customers, but is not usually integrated with the system model. The use case and domain models are typically ambiguous and incomplete, and its consistency cannot be automatically validated, because most of the constraints are specified in textual natural language. This kind of models is mainly used to reason about the system being built, and for sharing information between the project team members and other stakeholders.

Model-driven development approaches, like OMG's Model Driven Architecture (MDA), refocus the aim of modeling from human understandable to system recognizable [8], in order to allow the automatic verification, validation and transformation (including code generation) of system models. In this context, there is the need to improve the "quality" of the system model, i.e., its rigor and completeness, and also the consistency among sub-models and thus the global model integrity.

Our goal is to automatically generate an executable user interface (UI) prototype from an early system model (with an executable core), in order to allow the early validation of the system model and help in requirements elicitation and validation. The UI generated can be subject to usability and appearance improvements (without loosing the links to the underlying system model), and can also be used as a basis for subsequent system development.

For enhancing the preciseness of the model, OCL predicates will be used to formalize pre and post-conditions of use cases and domain classes' operations, and invariants of domain classes' attributes.

The UI generation process will take advantage of the OCL pre-conditions and invariants, defined both in the use case and domain model, to enhance the usability and behavior of the UI. On the other hand, the OCL post-conditions, if written in explicit form, can be used to allow model execution.

Thus, the main contribution of this work will be an approach for the automatic generation of functional user interfaces prototypes from early system models with minimum effort for model validation and requirements elicitation and validation purposes. The

start point are the system's domain and use case models enriched with OCL constraints. The approach will be focused on business applications and form based UIs.

In the next section, it is presented a state of the art survey on UI modeling and generation. Section 3 presents the main objectives of this research and the approach that will be taken. Sections 4 and 5 address the work plan and try to anticipate some expected results and establish a set of goals for publication. Finally, section 6 restates the research goals and approach, and draws some conclusions.

2. State of the Art

Modern software development processes typically use an iterative and incremental approach for developing software [7]. This allows the software engineers to cope with the ambiguities of the human language used for requirements elicitation. The user requirements that are about the system's functionality are then modeled in a form that may be more or less formal and rigorous. Typical types of models used nowadays are visual models (based on diagrams) and formal models (based on formal methods and notations with a mathematical foundation) [4, 11, 15].

Typical methodologies for modeling interactive applications use disparate views, or (sub)models, to capture different aspects of the domain (task model, dialogue model, abstract and concrete presentation models or application model) [13].

Most of existing approaches to UI generation require the specification of a UI model, like the ones studied by Pinheiro da Silva [13].

Some research has been made in order to model interactive systems using UML diagrams [14], but they also involve the full specification of the user interface.

As mentioned earlier, a typical approach to software engineering using UML starts by developing a sketch of the core system model by producing a structural or domain model, which models the system's domain classes, its attributes, relations and operations, and a functional or use case model, which models the user's intended operations to be accomplished on the system through its user interface. To test this core system model with the users and other stakeholders it is needed a user interface. Ideally, the UI for testing the core system model would be derived from the core system's specification.

There is also some research on deriving user interfaces from a model of the system core. In [10], Martínez *et al.* present a methodology for deriving UI from early requirements existing in an organization's business process model. Their approach follows a set of heuristics for extracting use cases and actors from the business process model. Each use case's normal and exceptional scenarios are then specified using message sequence charts enriched with UI related information. These UI enriched sequence diagrams are then used for automatically generating application forms and state transition diagrams for the interface objects and control objects present in the sequence diagrams.

Elkoutbi *et al.* [3] also approach UI generation by identifying usage scenarios. Their approach starts from a system domain structural model with OCL constraints and a use case model, but proceed by formalizing each use case through a set of UML collaboration diagrams, each corresponding to a use case scenario. Then, each collaboration diagram message is manually labeled with UI constraints (*inputData* and *outputData*) that identify the input and output message parameters for the UI. From the UI constraints it then automatically produces message constraints with UI widget information. Statechart diagrams are then derived from the UI labeled collaboration diagrams on a per use case basis. A statechart is created for each distinct class in a collaboration diagram. Then, state labeling and statechart integration are done incrementally, in order to obtain only one statechart per collaboration diagram, that is, per usage scenario. Elkoutbi's approach is then able to derive UI prototypes for every interface object defined in the class diagram.

In [12], Nunes uses activity diagrams to represent all scenarios of a given use case in only one model.

The use of OCL-like constraints to formalize use case specifications, although with testing purposes, is addressed by [16]. There, a new language is proposed (HCL – High-level Constraint Language) for formally specifying use cases' pre- and post-conditions. Each use case specified through HCL is a self contained specification, where input and output use case parameters are identified and related through pre- and post-conditions and any state variable that may be needed is defined locally. Each use case specification takes the form of a contract with no connection with other use cases or with the system state.

3. Research Objectives and Approach

We claim that it is possible to automatically generate a user interface with adequate quality to interactively validate the system's intended functionality, with minimum effort in the construction of the system model.

In our approach the system's domain structure is captured in UML class diagrams and the system's

functional aspects are captured using use case diagrams. For enhancing the rigor of the model and allow its subsequent automatic treatment, class diagrams are complemented with OCL formal statements that constrain the system state (by defining invariant conditions on the instance variables) and behavior (by defining pre- and post-conditions for each method); use cases are formalized with pre-/post-conditions in OCL, relating each use case with the system state.

Having pre-/post-conditions formalized for each use case, we claim that it is possible to generate a satisfactory user interface prototype without detailing individual usage scenarios.

One arising research question relates to the kind of UI that is possible to generate automatically from the system model. Our approach will first try to derive a default UI from the system domain model and only then will complement the domain model with a functional (use case) model, in an integrated manner, that allows for the automatic generation of the UI in a way that is closer to the user's intentions.

Another problem derives from the goal of executing a system model that is specified in a way that states what the system must do, but omits how the system will do it. In fact, OCL is a constraint language for writing Boolean expressions without side-effects. Nevertheless, it is possible to write post-conditions in such a way that they can be executed in order to provide a system state variables update. That is, a state transition in the underlying implicit system state machine. This way, the system model (prototype) can be animated through the generated user interface.

What is meant by "adequate quality" and how it is possible to reach an optimal balance between it and "minimum effort" is another question. We believe that, from a system model composed only of a rigorous specification of its structural and functional views, it is possible to derive a UI that is guided by the use cases, and so is close to the user perspective of the system. Such a generated UI is consistent with the system model and is complete in what respects to the system functionality. Aesthetic issues aren't addressed by the generation process, but we think that some usability issues can be enhanced. We claim that, from the use cases and classes' methods pre-conditions it is possible to derive a set of UI features that leverage the UI usability. Those characteristics include the provision of default values for mandatory UI fields, the suggestion of a set of system state dependent values for an input field, etc.

What metrics will be used to evaluate the "quality" of the generated UI and how is it possible to incorporate user feedback concerning UI aspects in the generated UI are another kind of concerns that must be addressed by the research work.

4. Current Work and Preliminary Results

The research work that has been done so far is mainly related to the literature review.

Current work is being done in the definition of a set of heuristics for UI generation. The first approach being established generates a UI from the system's structural model (domain classes diagram with constraints). In [2], a systematic approach to UI generation from domain classes complemented with class contracts in VDM++ is presented. The obtained user interface maps the system domain structure and the generated functionality is limited to the CRUD operations (Create, Read, Update, Delete), although it could be extended with properly stereotyped class methods.

The next step, according to the schedule in section 5, will be to automate those heuristics in a tool that shall automatically derive a default UI from the system's domain model. This can be done by deriving a UI model, in some UI modeling technique [13, 14], but we believe that if a UI description in a XML-based UI description language is generated it will be easier to execute the UI [5, 9].

Such UI derived only from the domain model is structural and constructive, as its functionality is typically aimed at populating the data structures (classes).

5. Work Plan and Implications

The PhD research work plan is scheduled to proceed as follows:

- **2006/07**:
  - Study the state of the art concerning Model-based User Interface Development and automatic generation of user interfaces from the core system model;
  - Define heuristics for automatic UI generation from domain structural models.
- **2007/08**:
  1st Semester:
  - Develop a tool for deriving a default UI from domain structural model according to the heuristics previously defined;
  - Define a set of recommendations for writing OCL post-condition constraints in UML models that facilitate the execution of the system core model through the generated UI;

- Develop or select a tool for "executing" post-conditions that are written according to the previous recommendations;

2nd Semester:
- Enhance the previous tool for allowing the animation (execution) of the system model through the generated UI;
- Establish an approach for deriving a UI from domain model and use cases with pre-/post-conditions.
- Develop a tool to automatically generate a UI from domain model and use case model;

- **2008/09**:
  1st Semester:
  - Define criteria to incorporate user feedback concerning UI aspects in the generated user interface;
  - Establish a set of UI evaluation metrics;
  - Test and evaluate the generated UI for a set of case studies;

  2nd Semester:
  - Write the PhD thesis report.

During the research work, we are planning to produce some published results. The goal is to publish, at least, once per semester, in the following topics:

- Heuristics for the automatic generation of CRUD UIs from structural domain models;
- Writing OCL post-condition constraints in a manner that facilitate the execution of the system core model;
- Deriving a state machine diagram from use cases pre-/post- conditions;
- A tool for the automatic generation of UIs from domain model and use case models.

6. Conclusions

The PhD research work envisioned in this paper aims at studying the automatic generation of a user interface from a system model that captures both the system's structure and the system functionality. The system structure is modeled by a UML class diagram complemented with OCL constraints. The class diagram shall model the system state (data structures) and also each class behaviour or responsibility (class methods).

The system functional characteristics, in what respect to the interaction with the user, are modeled by a UML use case model. Each use case will be further detailed by using pre- and post-conditions that link to the system state (specified by the UML class diagram). This way, there is a full integration between the two composing models, as use cases formal specifications are established over the structural model.

In the envisioned approach, a use-case may be understood as a user intended task on the system, like introduced by L. Constantine's essential use cases [1] and adopted by several requirements engineering methods [12, 6], or as a system *pack* of functionality, like defined by the Unified Process [7].

We believe that if each use case is understood as a user intended task on the system, i.e., an essential use-case, the generated UI will be closer to the user expectations and thus will have a higher degree of usability. In this case, subsequent iterations of user feedback may subdivide each essential use case, or user intended task, in further refined use cases, or sub-tasks, which, in turn, will have to be formalized using pre-/post-conditions. Indeed, this should be the normal iterative design process, and we claim that the envisioned approach will also respond adequately and that the generated UI will be further refined and closer to the user expectations.

As a way of leveraging the usability level of the generated UI, we also intend to use the use cases' pre-conditions and methods' pre-conditions to provide default values and lists of possible values for some of the input fields in the generated UI.

A set of metrics for evaluating the results obtained will need to be established. Results will be tested and evaluated using at least the case studies used in other approaches, such as the ATM machine of [3] and the Library System of [14].

7. References

[1] Constantine, L., "Users, Roles, and Personas", In *The Persona Lifecycle.* Eds. John Pruitt, Morgan-Kaufmann, 2006, chapter 8.

[2] Cruz, A. M. Rosado, "Deriving Default User Interfaces from Domain Contracts", *To appear in the 2nd Iberian Conference on Information Systems and Technologies* (CISTI 2007), Fernando Pessoa University, Porto, June 2007.

[3] Elkoutbi, M., I. Khriss, and R. Keller, "Automated Prototyping of User Interfaces Based on UML Scenarios", *Journal of Automated Software Engineering*, 13(1), Springer Science+Business Media B.V., January 2006, pp. 5-40.

[4] Fitzgerald, J., and P. Larsen, *Modeling Systems. Practical Tools and Techniques in Software Development*. Cambridge University Press, 1998.

[5] Forbrig, P., A. Dittmar, D. Reichart, and D. Sinnig, "From models to interactive systems tool support and

XIML". Proceedings of the First International Workshop on Making Model-based User Interface design practical: usable and open methods and tools (MBUI 2004), Funchal, Madeira, Portugal, 2004.

[6] Graham, Ian, *Requirements Engimeering and Rapid Developmment: An Object-oriented approach*, Addison-Wesley Longman, ACM Press, 1998.

[7] Jacobson, I., G. Booch, and J. Rumbaugh, *The Unified Software development Process*, Addison-Wesley, 1999.

[8] Kleppe, A., J. Warmer, and B. Wim, *MDA Explained: The Model Driven Architecture – Practice and Promise*, Addison-Wesley Professional, 2003.

[9] Kuo, Y., N. Shih, L. Tseng, and H.C. Hu, "Generating form-based user interfaces for XML vocabularies". *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering,* ACM Press, 2005, pp. 58-60.

[10] Martinez, A., H. Estrada, J. Sánchez, and O. Pastor, "From Early Requirements to User Interface Prototyping: A methodological approach", *Proceedings of the 17th IEEE International Conference on Automated Software Engineering* (ASE 2002), 2002, pp. 257-260.

[11] Meyer, B. "Dependable Software". In *Dependable Systems: Software, Computing, Networks*, Springer Berlin / Heidelberg, *Lecture Notes in Computer Science* vol. 4028, 2006, pp. 1-33.

[12] Nunes, N. Jardim, "Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach". PhD thesis, University of Madeira, 2001.

[13] Pinheiro da Silva, P. (2000). "User interface declarative models and development environments: A survey". In *Interactive Systems - Design, Specification, and Verification: 7th International Workshop, DSV-IS 2000, Limerick, Ireland, June 2000. Revised Papers*, Springer Berlin / Heidelberg, *Lecture Notes in Computer Science* vol. 1946, 2000, pp. 207–226.

[14] Pinheiro da Silva, P., "Object Modeling of Interactive Systems: The UMLi Approach". PhD thesis, Faculty of Science and Engineering, University of Manchester, 2002.

[15] Schoeller, B., T. Widmer, and B. Meyer, "Making specifications complete through models". In *Architecting Systems with Trustworthy Components*, Springer Berlin / Heidelberg, *Lecture Notes in Computer Science* vol. 3938, 2006, pp. 48-70.

[16] Shen, W. and S. Liu, "Formalization, Testing and Execution of a Use Case Diagram". In *Formal Methods and Software Engineering*, Springer Berlin / Heidelberg, *Lecture Notes in Computer Science* vol. 2885, 2003, pp. 68-85.