Towards Model Driven Engineering of Plastic User Interfaces

Jean-Sébastien Sottet
CLIPS-IMAG,
University of Grenoble, France
jean-sebastien.sottet@imag.fr

Gaëlle Calvary
CLIPS-IMAG,
University of Grenoble, France
gaelle.calvary@imag.fr

Jean-Marie Favre
LSR-IMAG
University of Grenoble, France
jean-marie.favre@imag.fr

ABSTRACT
Developing advanced User Interfaces (UI) is very challenging, in particular when many variants of a same UI are to be produced for different platforms. The development of *plastic user interfaces* is even more demanding. In Human Computer Interaction, *plasticity* denotes the capacity of a UI to withstand variations of the context of use while preserving usability. A context of use is a triplet < user, platform, environment >. Plasticity raises many issues for both the design and run time of UIs. This paper shows how Model Driven Engineering concepts can be used in this area.

1. INTRODUCTION

The need of *models* in Human Computer Interaction (HCI) has been recognized for long. Nevertheless full automatic generation of user interfaces (UI) rapidly shown its limits [1]. That does not mean that model-based techniques are not valuable, but that they have to be further explored with other objectives. First, traditionally limited to machine or human processed forward engineering, they have to be investigated in reverse and cross engineering. Secondly, models should live at run time instead of being limited to the design time.

Model Driven Engineering (MDE) [2] advocates the systematic use of "productive" models, that is, models that can be processed by the machine. Full engineering processes are described by explicit networks of models connected by explicit mappings and transformations [3]. This implies the systematic description of explicit metamodels, that is models of the modeling languages used to describe each model. Actually, most of the time the focus is set on transformation and traceability rather than on runtime productive models.

The core idea of this paper is to investigate Model Driven Engineering for the development and execution of plastic UIs. In HCI, plasticity denotes the capacity of a UI to withstand variations of context of use while preserving usability. A *context of use* refers to the triplet < user, platform, environment >. With ubiquitous computing, UIs are no longer confined in a unique desktop, but they can be distributed or migrated among a dynamic set of interaction resources. As a result, UIs must be mold dynamically in order to adapt gracefully to the new context of use. While product line approaches deal with the production of variants of UIs (e.g., a UI specifically crafted for PC or PDA or phone), plasticity is even more challenging coping with the change of the context of use. The UI can fully migrate from one platform to another one (e.g., from a PC to a PDA when the battery of the PC gets low); the UI may be redistributed among a set of platforms (e.g., a remote controller will migrate to a PDA) or the UI may stay on the current platform but be remold in order to accommodate to variations in the context of use (e.g., luminosity level). Adaptation should be done in an opportunistic manner for preserving usability.

While code-centric approaches might be suited for the development of simple and single UIs, they simply fail for developing plastic UIs. Many facets of the interactive system, in particular its plasticity domain (i.e., the set of contexts of use it is able to cover) must be modeled explicitly. This is what Model Driven Engineering is suited for.

Figure 1 shows an overview of a MDE framework based on the European CAMELEON project [5] addressing the development of Plastic UIs. A (simplified) version of this framework will be shortly described in this paper. Each package on the first line represents a metamodel (**M2** level of the meta pyramid [4]). As the reader can see, many facets have to be made explicit (e.g., the User, the Platform, …). The models representing a specific interactive system fit in the **M1** level. In a given column, all models *conform to* [4] the same metamodel. For clarity, mappings and transformations are not shown in the figure.

The paper is threefold. Section 2 focuses on the right part of Figure 1, that is the development of "rigid" (i.e., not plastic) UIs. Specific requirements for plasticity are described in Section 3 (left part of Figure 1). Finally Section 4 concludes the paper.

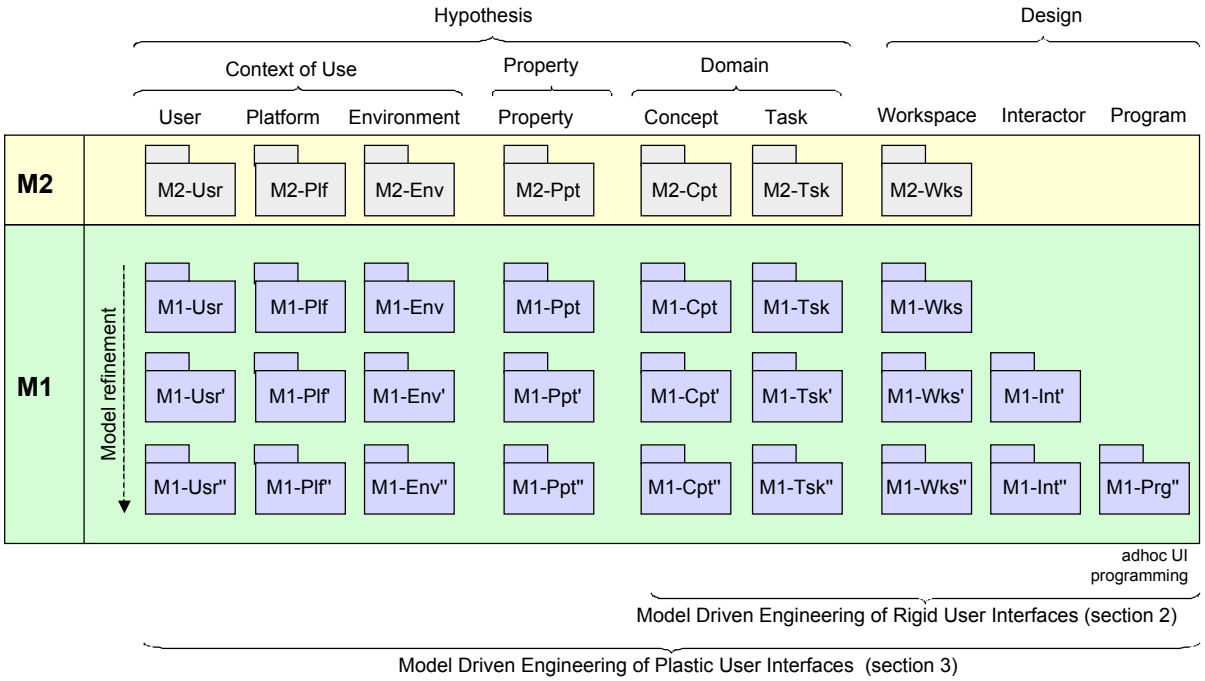

Figure 1. A Model Driven Engineering Framework for the Development of UIs

2. MDE OF RIGID USER INTERFACES

Before addressing the specific requirements for plasticity, let's see how Model Driven Engineering can be used for the development of classical rigid UIs by opposition to plastic UIs. This implies the use of (1) a development process (or better said a family of development processes), (2) a set of metamodels and (3) a set of mappings and transformations. A more detailed discussion on this approach can be found in [6].

2.1 Development Process

Current industrial practices are still mostly code-centric (right bottom of Figure 1). Conversely, Model Driven Engineering processes are based on successive refinements of models, with the integration of new information at each step [5]. Teresa [11] and UsiXML [10] exemplify forward engineering processes. They consist in reifying models step by step from an entry point [16] (in most cases, the task model) until reaching the final running program. Typical development processes start from domain models such as user tasks and concepts models. Then, based on this knowledge and design choices, the UI is designed in terms of workspaces, interactors and finally program elements dependent on specific platforms and libraries. In practice, processes are iterative rather than straightforward and the number of steps may vary. Moreover as suggested by Figure 1, very often it is necessary to adapt an initial model to the various constraints introduced at each level. For instance an initial task model (M1-Tsk) might be tuned into another one (M1-Tsk') to take into account constraints related to concrete library toolkits (M1-Tsk'').

2.2 (Meta)Models for Rigid UI Engineering

Metamodels are keys to Model Driven Engineering. To be productive, each model must be accompanied with a precise and explicit metamodel (otherwise the model cannot be interpreted and transformed by a machine). The development of classical rigid UIs implies five metamodels [6]. A simplified backbone composed of four metamodels is given in Figure 2. The mappings between metamodels will be explained in the next section.

- **Task**. A task describes how a user's objective can be reached by following a given procedure. From a MDE point of view, task models are trees made of binary and unary operators.
- **Concept**. This model describes the domain of discourse. Typically such models can be described using UML class diagrams. These are usually referred as Domain Model in Software Engineering and Product Line communities.
- **Workspace.** Also referred as Abstract User Interface [5], this model describes the network of workspaces in which tasks are performed.
- **Interactor.** Also referred as Concrete User Interface in [5], this model can still be described in terms of abstract Interactors. Various levels of abstraction can be considered here.
- **Program**. The last "models" are direct abstractions of programs and/or other implementation techniques. Such models are Platform Specific Models (PSM) in the MDE jargon. They just represent the actual implementation of the UI.

Obviously, all models expressed with these metamodels should be connected together because they just represent different points of view on the same UI. Mappings and transformations are briefly discussed in the next section.

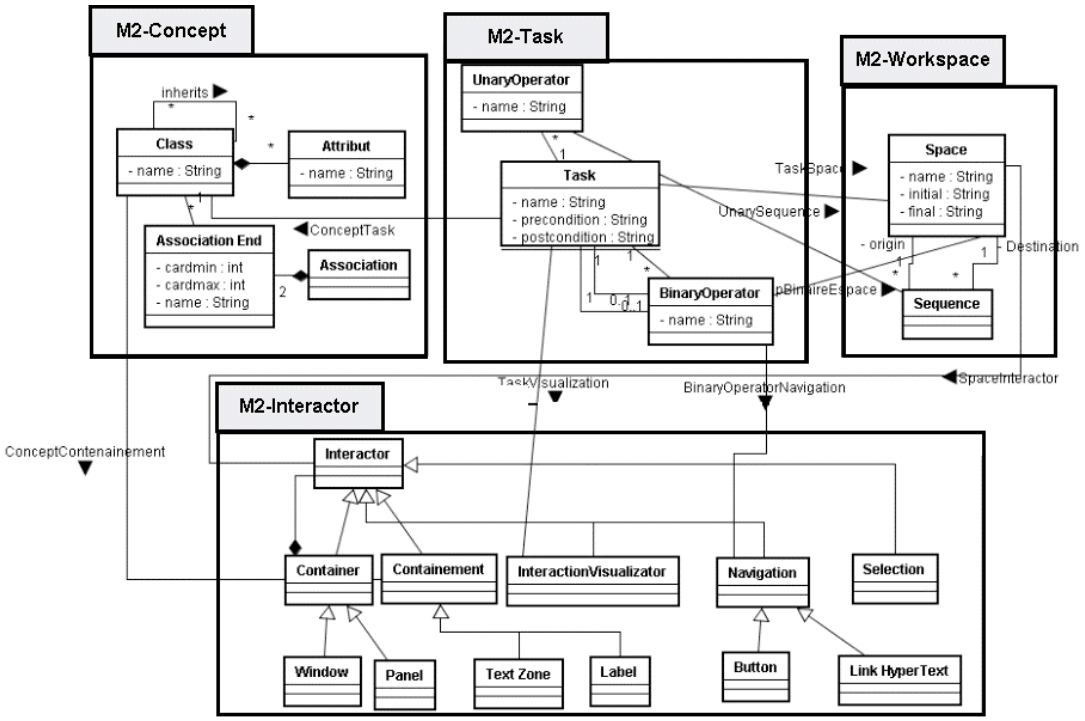

Figure 2. Mappings Between 4 Metamodels for Rigid User Interface Engineering (simplified view)

2.3 Mapping and Transformations

With metamodels, mappings and transformations are the cornerstones of MDE. Without them, all the models would be isolated. Conversely, the idea is to incrementally transform abstract models at high level of abstraction (Platform Independent Models, PIM in MDE jargon) into Platform Specific Models. This is obviously an over simplification. Monolithic code generators are not suited for advanced user interfaces. MDE approaches are based on libraries of small composable and extensible transformations. The designer selects and if necessary tunes the appropriate transformations. If no transformation is available, a new one can be written thanks to transformation languages. It is then added to a library for further use. In this way, expertise can be captured and packaged into transformation libraries.

Quite often transformation engines are associated to specific modeling environments based on a given set of metamodels for UI development. This is the case for instance for TransformiXML [12] in the UsiXML environment. While this kind of approaches is worth, it is specific to UI development. It does not cover the whole software engineering process.

The core idea of our approach is to use generic Model Driven Engineering techniques and extensive libraries of metamodels. This approach is being investigated in the Zooomm project [2]. While emerging standards for expressing MDE transformations are under active development (e.g. QVT), we investigate [6] the appropriateness of the ATL generic MDE transformation language [12] for plasticity. The following piece of code is an example of transformation written in ATL. It describes very simple rules transforming Tasks into Workspaces:

- The rule *TaskToSpace* creates one workspace *w* per user task *t*. The workspace takes the name of the task;
- The rule *OrOperatorToSequence* applies to the task model. It transforms all OR operators *o* between two user tasks (*o.RightTask*) into two sequence operators (from o.*motherTask* to *o.leftTask* and *o.leftTask* to *o.rightTask*).

```
module M2TaskToM2Workspace {
  from M1Task : M2Task
  to   M1Workspace : M2Workspace

  -- One workspace for each task
  rule TaskToSpace {
    from t : M2Task!Task
    to w : M2Workspace!Space (
      name <- t.name
    )
  }

  -- OrOperator
  rule OrOperatorToSequence{
    from o : M2Task!BinaryOperator (
      o.name = "or"
    )
    to leftSequence : M2Workspace!Sequence (
      origin<- [Task2Space.e]o.motherTask,
              destination<-[Task2Space.e]o.rightTask )
  ...
```

3. TOWARDS MDE OF PLASTIC UI

As shown on the left of Figure 1, additional models are necessary for the engineering of plastic UIs.

3.1 Additional (Meta)Models for Plastic UI

For plasticity, we need additional metamodels for capturing both the context of use and the usability of the interactive systems.

- **User.** This model represents the end-user of the interactive system. It may capture general information (e.g., age, gender), the skill level of the end-user (e.g. Rasmussen [18]) in both computer science and in the applicative domain.
- **Environment.** This model represents the physical (e.g., the noise level), social conditions where the interaction takes

place. From an engineering perspective, the environment can be modeled as a graph of contexts and situations [17]. Additionally the model could contain social rules such as "switch off the volume in a train", etc.

- **Platform**. This model exhibits the hardware and software platform sustaining the interaction. It is fundamental when considering redistributable UIs. The simple metamodel below describes the hardware core elements of such a system [9]. Platform modeling is an important issue in Model Driven Engineering, and expertise from this domain could be reused.

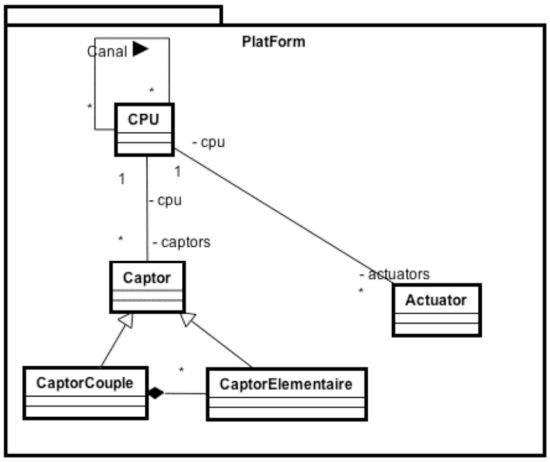

Figure 3 Simplified hardware platform metamodel for UI

- **Usability.** While some elements could be modeled in a single package, existing frameworks (e.g., Scapin & Bastien [7], IFIP [8]) show that many elements could be attached on relations between models. For instance, the IFIP *observability* property may be seen as a link between concepts and concrete UI elements. If the concept is not mapped on an UI element then it is not observable.

3.2 Runtime Adaptation

While models are mostly used at design time in MDE and product-line engineering, plasticity may rely on an extensive use of models at run-time. Moreover, all abstraction levels and traceability links are required during execution to compute the adaptation. For instance, moving a selection task from a PC to a PDA is easier when reasoning at the task level rather than at the implementation level. The corresponding interactors can be simply found by following the mapping links [9]. Then for a migration, a new set of program elements should be computed on the target platform, and this through traceability links. Sometime adaptation requires knowledge about the heuristics that were used former design choices. For instance, some concepts could have been suppressed for targeting small devices. These concepts should be recovered when migrating to a more powerful device.

4. CONCLUSION

The use of models is not new in User Interface development. First attempts have been directed towards full automatic generation of UI code. Experience has shown that the resulting UIs were usually poor in terms of usability. Moreover, the use of quite monolithic code generators make it impossible to customize the interface when needed and specific heuristics based on application domains could not be integrated. Finally, existing environments were not designed with interoperability in mind.

The approach presented in this paper is quite different. Instead on focusing on the user interface only, general Model Driven Engineering techniques have been used. The key idea is to merge experiences from both the MDE and the UI development communities. Instead of developing specific model based tools such as transformation languages for the development of UI, reusing emerging MDE technologies is promising. First versions might not be fully suited to UI development specificities, but if this is the case, this would lead to new requirements for MDE. The framework presented here is used both at design time and run-time. This last point is quite innovative with respect to rational MDE applications that consider platform migration as a quite heavy development process.

5. ACKNOWLEDGMENTS

This work has been supported by the SIMILAR European Network of Excellence.

6. REFERENCES

[1] Myers B., Hudson S.E., Pausch R. "Past, Present, and Future of User Interface Software Tools", *Transactions on Computer-Human Interaction (TOCHI), Vol 7, Issue 1*, 2000

[2] Planet MDE, "A Web Portal for the Model Driven Engineering Community" http://planetmde.org

[3] Favre J.M., "Foundations of Model (Driven) (Reverse) Engineering", Dagsthul Seminar on Language Engineering for Model Driven Development, DROPS, http://drops.dagstuhl.de/portals/04101, 2004

[4] Favre J.M., "Foundations of the Meta-pyramids: Languages and Metamodels", DROPS, http://drops.dagstuhl.de/portals/04101, 2004

[5] Calvary G., Coutaz J. Thevenin, D. Limbourg, Q., Bouillon, L., Vanderdonckt J. "A Unifying Reference Framework for Multi-Target User Interfaces, *Interacting With Computers*, 2003

[6] Sottet J.S., Calvary G., Favre J.M., "Ingénierie de l'Interaction Homme-Machine Dirigée par les Modèles", *IDM05, Paris,* 2005.

[7] Scapin D., Bastien, C.H., "Ergonomic Criterias for Evaluating the Ergonomic Quality Interactive Systems." *Behaviour and Information Technologies, Vol 16,* 1997

[8] Abowd G., Coutaz J., Nigay L., "Structuring the Space of Interactive System Properties", *Proceeding of the IFIP*, 1992.

[9] Demeure A., Calvary G., Sottet J.S., Vanderdonkt, J., "A Reference Model for Distributed User Interfaces", *TAMODIA'2005*.

[10] Limbourg Q., Vanderdonckt J., Michotte, B., Bouillon, L. Florins, M. Trevisan D., "UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces", *AVI 2004,* Gallipoli, 2004.

[11] Mori G., Paternò F., Santoro C. "Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions" *IEEE Transactions on Software Engineering,* August 2004.

[12] Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Lopez-Jaquero, V., "UsiXML: a Language Supporting Multi-Path Development of User Interfaces", *9th IFIP Working Conference on Engineering for Human-Computer Interaction EHCI 2004*

[13] Nunes N., Cunha J.F, "Toward Flexible Automatic Generation of User-Interface via UML and XMI", *5th Workshop Iberoamericano de Ingenieria de Requisitos y Ambientes Software, IDEAS 2002*

[14] Zooomm, "Zooomm, The International ZOO Of MetaModels, Schemas and Grammar for Software Engineering", http://zooomm.org

[15] Mori, G., Paternò, F., Santoro, C. "Tool Support for Designing Nomadic Applications" *Proceedings ACM IUI'03*, Miami, pp.141 ACM press

[16] Limbourg, Q. "Multi-path Development of User Interfaces", *PhD of University of Louvain La Neuve*, Belgium, 2004

[17] J. Crowley, J. Coutaz, G. Rey, P. Reignier Perceptual Components e for Context-Aware *Computing, UbiComp 2002*:, Göteburg, Sweden Sept./Oct. 2002

[18] Rasmussen, J. (1983). Skills, rules, and knowledge; signals, signs, and symbols; and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-13,* 257-266