Adding Flexibility in the Model-Driven Engineering of User Interfaces

Nathalie Aquino
Centro de Investigación en Métodos de Producción de Software, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
+34 963877007 ext. 83534
naquino@pros.upv.es – http://www.pros.upv.es

ABSTRACT
Model-based user interface (UI) development environments are aimed at generating one or many UIs from a single model or a family of models. Model-driven engineering (MDE) of UIs is assumed to be superior to those environments since they make the UI design knowledge visible, explicit, and external, for instance as model-to-model transformations and model-to-code compilation rules. These transformations and rules are often considered inflexible, complex to express, and hard to develop by UI designers and developers who are not necessarily experts in MDE. In order to overcome these shortcomings, this work introduces the concept of transformation profile that consists of two definitions: model mappings, which connect source and target models in a flexible way, and transformation templates, which gather high-level parameters to apply to transformations. This work applies these concepts in a general-purpose method for MDE of information systems. Transformation profiles can be effectively and efficiently used in any circumstances in which transformation knowledge needs to be modified by non-experts, and flexibility, modifiability, and customization are required.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Computer-aided software engineering (CASE), User interfaces*. H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces (GUI), Style guides*.

General Terms: Design, Human Factors.

Keywords: Model-driven engineering, user interface, model transformation, profile, template.

1. INTRODUCTION
Model-based user interface development environments (MB-UIDEs) provide a context within which user interface (UI) declarative models that describe a UI and aspects involved in it (e.g., task, domain, context of use) can be constructed and related, as part of the UI design process [5]. From these models, one or many UIs can be (semi-)automatically generated. In a similar way, the model-driven engineering (MDE) of UIs implies the definition of UI declarative models from which a final UI is produced. However, in this case the UI design knowledge and presentation guidelines can be made visible, explicit, and external by means of model-to-model (M2M) transformations and model-to-code (M2C) compilation rules.

MB-UIDEs and MDE of UIs provide a way in which UI designers and developers can design and implement UIs in a professional and systematic way, more easily than when using traditional UI development tools [5]. However, although MB-UIDEs and MDE of UIs incorporate advantages and automation in the UI development process, at the same time, they create a new problem: how could UI designers interfere in an automatic UI development process when they need to develop UIs with characteristics that go beyond the scope of the process they use? [5]. First of all, why would UI designers need to interfere in the automatic process they use? Basically, they will need to interfere if the process is not powerful enough to generate the kind of UIs they need. In fact, one identified challenge in MDE of UIs is the "*need for powerful transformation and rendering engines"* [14]. Most currently available transformation and rendering engines do not take advantage of the full power of the languages in which they generate the UI code and, hence, they are not powerful enough to produce attractive UIs in which it is not necessary to make changes. Secondly, why would UI designers need to develop UIs with characteristics that go beyond the scope of an automatic process, even if they use a powerful one? This can be related with end-user requirements such as "*the desire to keep control over an application, the need to be compliant with a particular style guide, the need to cope with user preferences that were not considered in the MDE approach*" [14]. And thirdly, how do UI designers interfere in current UI design processes? On the one hand, there are MB-UIDEs which have their design knowledge and presentation guidelines implicit and fixed in compilers that perform the transformations. This results in the generation of predetermined UIs, all of them looking alike. In these cases, there is a lack of flexibility for UI designers to customize the generated UIs, since for that purpose they must manually tweak the generated UI code. Manual tweaking can lead to problems understanding and modifying the generated code, to create inconsistencies between the UI and its model, and to endanger quality features guaranteed by an automated process such as absence of errors in the code. Furthermore, if a UI that has been generated automatically from a model is manually tweaked and, later, its model is changed, the re-generated UI will change according to the new model, but manual tweaks made previously will be lost. On the other hand, there also exist approaches for MDE of UIs which have their design knowledge and presentation guidelines explicit and expressed, either with mapping and transformation models or with model transformation languages. In these cases, the customization can be made previously to the UI generation. However, UI designers must define the new transformation options they need, or edit existing

EICS'09, July15–17, 2009, Pittsburgh, Pennsylvania, USA.

ones. That complex process, which requires some background in MDE, such as M2M transformations, graph transformations, term rewriting, transformations rules, etc., is more oriented to model transformation specialists than to UI designers.

Considering the current situation of automated UI design approaches, it can be said that the complete support to end-user requirements related to UIs is a problem which has not been totally resolved. It can also be affirmed that UI designers need more flexible approaches for the automatic generation of UIs from models. In particular, they need that approaches incorporate flexible mechanisms to specify concrete aspects of UIs taking into account the context of use of the UIs: end-user preferences and characteristics, hardware and software platforms, and environment [4]. This puts forward an interesting work in which research interests of the Software Engineering and Human-Computer Interaction communities converge.

2. TRANSFORMATION PROFILES

This work proposes to use *Transformation Profiles* in order to support flexibility, modifiability, and customization in automated UI development processes. The transformation profile approach has been introduced in [2].

A transformation profile is made up of a set of *Model Mappings* and a *Transformation Template*. On the one hand, model mappings connect initial and target UI models in a flexible way. They are intended to specify the structure and layout of the UI, as well as widgets selection. Model mappings provide flexibility in UI development processes since they can be used to externalize the design knowledge in approaches that have it implicit in transformation engines. Furthermore, the externalized design knowledge can be customized editing model mappings and, later, can be reused in similar UI development projects. On the other hand, a transformation template gathers *parameters* that specify details of the structure, layout and style of UIs. Each parameter corresponds to a *parameter type* which determines the UI elements on which the parameter can be *applied* as well as the UI elements which can be *affected* by the parameter. A parameter could affect the same UI element in which it is applied or could affect a UI element which is contained in another UI element in which it is applied. A parameter type also has a *priority* which is used to resolve conflicts when two or more parameters affect a same UI element causing the same effect. A parameter type has an associated *data type* or an enumeration of *possible values* as well as a *default value*. Each of these possible values can be assigned with *usability guidelines,* which are intended to help UI designers to choose appropriate values in different circumstances, and with *importance level* (IL) and *development cost* (DC) estimations, which can be used to decide which possible values to implement in different contexts of use as well as the implementation order. Parameter types with an associated data type can also be assigned with IL and DC estimations. In the case of parameter types with an enumeration of possible values the IL and DC are obtained averaging their possible values estimations. Parameter types can be related to single attributes of UI elements (e.g., colours, font types), or to a group of attributes of one or more elements, or to the elements themselves and relations among them (e.g., the specification of the widgets to be used, layout options, dialogue style, location of objects, alignment of elements). Parameters can overwrite the mappings defined in model mappings. As well as model mappings, transformation templates can be used to externalize, customize and reuse the UI design knowledge and presentation guidelines, but it is also intended to make it easier for UI designers to specify the transformations, since the selection of values for parameters is a simpler process than the modification of mappings.

Different transformation profiles can be defined to address different hardware and software platforms and end-user characteristics and preferences.

Figure 1 illustrates the use of a transformation profile when it is applied to an AUI (Abstract User Interface [4]) to CUI (Concrete User Interface [4]) model transformation. A transformation engine takes as input an AUI model and a transformation profile. The transformation profile provides the rules that specify how to transform the AUI model to the CUI model.

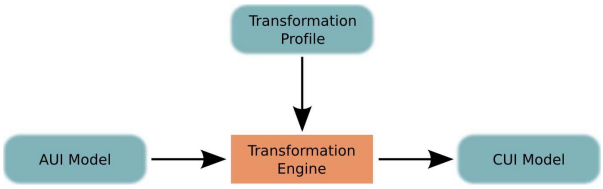


Figure 1. AUI model to CUI model transformation using a transformation profile

2.1 Transformation Profiles and OO-Method

The aim of this work is to define an approach for adding flexibility in automated UI development processes. The approach must be general enough to be applied in a wide range of approaches for UI development, and it should be particularly applicable to OO-Method. OO-Method [12] is an object-oriented method which allows the automatic generation of complete software applications (persistence, business logic, and presentation layers) from conceptual models. OO-Method uses a *presentation model* (PM) for specifying a UI in an abstract way. The OO-Method PM is based on a set of UI patterns which are defined in [10]. *OLIVANOVA The Programming Machine* is a commercial tool developed by CARE Technologies (http://www.care-t.com) which supports OO-Method. Applications can be generated to run in different software platforms (C# running on .NET, EJB/JavaServer Faces running on J2EE, among other options). A model compiler is implemented for each supported software platform. Currently, most of the logic that guides UI model transformations is implicit in *OLIVANOVA* compilers. Furthermore, if the generated applications are going to be used in different hardware platforms (e.g., desktop, handheld platforms) there is not a mechanism that allows generating the UIs in a way that takes advantages or avoids problems in the different platforms. Therefore, the application of the transformation profiles approach could be convenient.

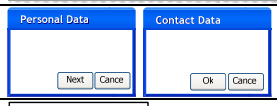
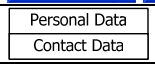
Two of the UI patterns defined in the OO-Method PM will be considered here in order to illustrate some concepts related to transformation templates: the *Service Interaction Unit* (SIU) and the *Argument Grouping*. A SIU is used for specifying the presentation of a service that modifies an object, their attributes and relationships. An Argument Grouping can be used with a SIU in order to arrange the input arguments of the service in groups and to establish the order in which groups should appear (more details about SIU and Argument Grouping can be found in [10]). Table 1 represents a subset of a mapping model that maps OO-Method UI patterns (source column) to UsiXML cuiModel [9] elements (target column).

Table 1. Mapping model (subset) for PM to UsiXML cuiModel transformations

Source	Target
SIU	*window* that contains a *border box* which encloses a *top box* and a *bottom box*. The *top box* contains a vertical-oriented *box*. The *bottom box* contains a right-aligned *flow box* with OK and Cancel *buttons*
Argument Grouping	*group box*

The mapping which relates Argument Grouping with group box could alternatively be expressed as a parameter in a transformation template. Table 2 illustrates a parameter type named *Grouping Layout* which defines the characteristics and possible values of parameters that allow selecting the type of container that will be used to group input arguments of a service.

Table 2. Grouping Layout parameter type

Parameter Type			
Name	*Grouping Layout*		
Priority	Medium		
Affects to	*Argument Grouping*		
Applies to	*Argument Grouping, SIU*		
IL	4		
DC	2.5		
Possible Values Enumeration			
Value	Graphical description	IL	DC
group box (default value)		5	2
tabbed dialogue box		5	2
wizard		4	3
accordion		2	3

Considering that currently OO-Method/*OLIVANOVA* generates UIs directly from the abstract PM, the incorporation of the transformation profiles approach implies the following tasks:

- The specification/selection of the semantic, syntax and stylistic for CUI models. A CUI model will constitute an intermediate step between the abstract OO-Method PM representation and the generated UI code. The UsiXML cuiModel has been selected for this purpose.
- The specification/selection of the semantic, syntax and stylistic for model mappings. The UsiXML mappingModel [9] (or a slightly modified version) is being considered for this purpose.
- The specification of semantic, syntax and stylistic for transformation templates. Transformation templates and all their related concepts have been clearly described with meta-meta-models and meta-models (which define semantic) [1].
- The implementation/selection of tools for creating and editing model mappings (future work).
- The implementation of tools for creating and editing transformation templates. A prototype has already been implemented [1].
- The externalization of the model mappings which are currently implicit in *OLIVANOVA* (ongoing work).
- The definition of alternative model mappings (future work).
- The definition of a meaningful set of parameters. A catalogue of parameters has been defined for the OO-Method PM [1]. More parameters could be added to the catalogue. Parameters for the UsiXML cuiModel should also be defined.
- The implementation of an OO-Method PM to UsiXML cuiModel compiler (future work) which takes as input a transformation profile.
- The implementation of a UsiXML cuiModel to UI code compiler (future work). This compiler can also take as input a transformation profile.

All these tasks will be carried out according to a design-science research approach [8] and the results will be empirically evaluated.

Recently, *OLIVANOVA* compilers have already adopted the parameter concept but have implemented it directly in the transformation from the OO-Method PM to UI code, without using a CUI Model. Currently, only a small set of parameters is implemented for each supported software platform and the parameters set is not the same for different software platforms. *OLIVANOVA* compilers have also incorporated a user preference configuration file that allows end-users to customize the displayed information and the position of certain scenarios. Most of those end-user options could also be considered and implemented as parameters. This would help to homogenize all parameterizable information.

This work is also interested in that the OO-Method/*OLIVANOVA* approach, enhanced with transformation profiles, can generate multi-device/platform UIs with good usability. Currently, an empirical study (end-user testing and heuristic evaluation) is being conducted in order to evaluate the usability of UIs generated with the original OO-Method/*OLIVANOVA* approach, considering different software platforms (C# running on .NET and EJB/JSF running on J2EE) and using different hardware platforms and devices (desktop platform, wall screen, and handheld platform). It is hoped that the results of the study help in identifying alternative model mappings, as well as parameters that result in the generation of more usable multi-device/platform UIs.

3. RELATED WORK

Some software tools support a transformation-based approach for generating a UI but the transformations are not made explicit (e.g., Teallach [7], TERESA [11]). With respect to these approaches, this proposal has the advantage that design knowledge and presentation guidelines are externalized and, therefore, are customizable and reusable in UI development projects with similar characteristics. Other software tools support explicit transformations. TransformiXML [9] interprets mappings written in UsiXML and converts them into graph transformations. Furthermore, different model transformation languages have been applied in automated UI development processes. Surveys of some of them are presented in [6, 13]. With respect to these approaches, this proposal has the advantage that it incorporates a parameterization process which is much more affordable for UI designers. There are also other software tools that support a template-based approach (e.g., Genova

- http://www.esito.no/, CSS [3]). With respect to these approaches, this proposal has the advantage that it does not depend on a particular UI development method but it can be used with different ones. Furthermore, it can guide transformations to different hardware and software platforms as well as to different end-user characteristics and preferences. Parameters of a transformation template are not limited to modifying widget properties, but they can also specify the structure and layout of the UIs. Parameters are not tied to programming or mark-up languages and new parameter types can be added when needed.

This proposal is also applicable to model transformations in which a UI model based on UI patterns is involved. To the best of our knowledge, no existing work today provides both a transformation-based approach and a template-based approach in a single and unified way of developing UIs. This combination combines the powerfulness of the first approach with the flexibility of the second.

4. CONCLUSION

The aim of this work is to provide a flexible way for UI designers to interfere in an automatic UI development process in order to customize the UI generation. For that purpose, the transformation profile approach is introduced. It is expected that following this approach more flexible and powerful transformation and rendering engines could be implemented. It is also expected that this approach results in a more affordable way for UI designers to specify UI transformations according to different end-user characteristics and preferences, and different software and hardware platforms. Furthermore, the transformation profile notion could be used in any M2M transformation or M2C compilation which needs to be customized by non-experts.

5. ACKNOWLEDGMENTS

This work has been developed with the support of MEC under the project SESAMO (TIN2007-62894), and by GVA under grant BFPI/2008/209.

6. REFERENCES

[1] N. Aquino. Plantillas de Transformación. Añadiendo Flexibilidad a las Transformaciones de Modelos de Interfaces de Usuario. Master's thesis, Universidad Politécnica de Valencia, 2008.

[2] N. Aquino, J. Vanderdonckt, F. Valverde, and O. Pastor. Using Profiles to Support Model Transformations in the Model-Driven Development of User Interfaces. In V. Lopez Jaquero, F. Montero Simarro, J. Molina Masso, and J. Vanderdonckt, Eds., *Computer-Aided Design of User Interfaces VI, Proc. of 7th Int. Conf. on Computer-Aided Design of User Interfaces CADUI2008, (June 11-13, 2008, Albacete, Spain)*, Springer, 2008.

[3] B. Bos, T. Çelik, H. W. Lie, and I. Hickson. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Technical report, World Wide Web Consortium (W3C), July 2007.

[4] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.

[5] P. P. da Silva. User Interface Declarative Models and Development Environments: A Survey. In *DSV-IS*, pp. 207–226, 2000.

[6] J. M. Gonzalez, A. Stanciulescu, J. Vanderdonckt, J.-P. Delacre, and M. Winckler. A Comparative Analysis of Transformation Engines for User Interface Development. In N. Koch, G.-J. Houben, and A. Vallecillo, Eds., *Proc. of 4th Int. Workshop on Model-Driven Web Engineering MDWE'2008 (Toulouse, 1 October 2008)*, vol. 389 of *CEUR Workshop Proceedings*, pp. 16–30, 2008.

[7] T. Griffiths, P. J. Barclay, N. W. Paton, J. McKirdy, J. B. Kennedy, P. D. Gray, R. Cooper, C. A. Goble, and P. P. da Silva. Teallach: a Model-Based User Interface Development Environment for Object Databases. *Interacting with Computers*, 14(1):31–68, 2001.

[8] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28(1), 2004.

[9] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V. López-Jaquero. USIXML: A Language Supporting Multi-path Development of User Interfaces. In R. Bastide, P. A. Palanque, and J. Roth, Eds., *EHCI/DS-VIS*, vol. 3425 of *LNCS*, pp. 200–220. Springer, 2004.

[10] P. J. Molina, S. Meliá, and O. Pastor. Just-UI : A User Interface Specification Model. In C. Kolski and J. Vanderdonckt, Eds., *CADUI*, pp. 63–74. Kluwer, 2002.

[11] G. Mori, F. Paterno, and C. Santoro. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Trans. Software Eng.*, 30(8):507–520, 2004.

[12] O. Pastor and J. C. Molina. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[13] R. Schaefer. A Survey on Transformation Tools for Model Based User Interface Development. In J. A. Jacko, editor, *HCI (1)*, vol. 4550 of *LNCS*, pp. 1178–1187. Springer, 2007.

[14] J. Vanderdonckt. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In S. Buraga and I. Juvina, Eds., *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008, (Iasi, 18-19 September 2008)*, pp. 1–10. Matrix ROM, Bucarest, 2008.