Using Profiles to Support Model Transformations in the Model-Driven Development of User Interfaces[1]

Nathalie Aquino[1], Jean Vanderdonckt[2], Francisco Valverde[1], Oscar Pastor[1]

[1]Department of Information Systems and Computation, Valencia University of Technology,
Camino de Vera s/n. 46022 Valencia (Spain)
{naquino, fvalverde, opastor}@dsic.upv.es – http://oomethod.dsic.upv.es/
[2]Université catholique de Louvain, Louvain School of Management (LSM)
Place des Doyens, 1 – B-1348, Louvain-la-Neuve, Belgium
E-mail: jean.vanderdonckt@uclouvain.be - Web: http://www.isys.ucl.ac.be/bchi

Abstract The model-driven User Interface (UI) development life cycle usually evolves from high-level models, which represent abstract UI concepts, to concrete models, which are more related to the UI implementation details, until the final UI is generated. This process is based on a set of model-to-model and model-to-code transformations. Several industrial tools have applied this approach in order to generate the UI. However, these model transformations are mainly fixed and are not always the best solution for a specific UI. In this work, the notion of Transformation Profile is introduced to better specify the model-to-model transformations. A Transformation Profile is made up of a set of predefined Model Mappings and a Transformation Template. The mappings connect initial and target UI models in a flexible way, whereas the Transformation Template gathers high-level parameters to apply to the transformation. As a consequence, a Transformation Profile enables designers to define parameterized transformations that could be reused for another UI development project.

1 Introduction

The Cameleon Reference Framework [1] defines a MDE-compliant (Model-Driven Engineering [13]) development life cycle for multi-target User Interfaces (UIs) and structures it into four levels of abstraction: *Task and Concepts* to describe tasks and domain-oriented concepts; *Abstract User Interface* (AUI), to express a UI in terms of Abstract Interaction Objects (AIOs) in a way that is independent from the interactors available in the targets; *Concrete User Interface* (CUI), to concretize the AIOs of an AUI into Concrete Interaction Objects (CIOs)

[1] This work has been developed with the support of MEC and MITyC under the projects SESAMO (TIN2007-62894) and PISA (FIT-340000-2007-110).

which are independent from a specific toolkit; and *Final User Interface* (FIU), the UI code in any programming or mark-up language.

This work focuses on the transformation from an AUI model to a CUI model. On the one hand, there are approaches where transformation rules are implicit in the transformation tools, resulting in a lack of flexibility to customize transformations [12,4]. On the other hand, typically there are similarities and differences among UI development projects. Therefore, it is not a reasonable approach to define the transformation rules for the AUI to CUI model transformation each time for each project, nor is it reasonable to use the same transformation rules for every UI development project.

The main purpose of this work is to optimize AUI to CUI model transformations. To achieve this goal, Transformation Profiles are introduced as a mechanism to externalize and customize the AUI to CUI model transformations and to re-use knowledge between different UI development projects. A Transformation Profile is composed of a set of Model Mappings and a Transformation Template. The Model Mappings specify how to concretize an AIO into a CIO. Therefore, the connections between the AUI model and the CUI model are externalized from the tools that perform the transformations and can be customized according to computing platforms and users. The Transformation Template parameterizes the transformation with high-level parameters that can be applied in two dimensions: UI fragments or UI patterns.

The rest of the paper is organized as follows: Section 2 presents Transformation Profiles, Model Mappings and Transformation Templates. Section 3 presents a case study with a practical application of the Transformation Profile in the generation process of UIs in OO-Method, a software development method. In the case study, the OO-Method Presentation Model plays the role of the AUI model, and the UsiXML CUI Model plays the role of the CUI model. In Section 4, the AUI to CUI model transformations of others MDE-compliant UI development methods are analyzed. Finally, Section 5 presents some conclusion.

2 Introducing the Transformation Profile Approach

In a MDA-compliant (Model-Driven Architecture [9]) UI development process, an AUI model is transformed to one or more CUI models. This transformation is based on mappings from elements of the AUI model to elements of the CUI model. The mapping problem has been defined as the difficulty of linking abstract and concrete elements in a UI model. This problem has been identified by Puerta and Eisenstein [12] as a non-trivial one. One of the main issues raised by the mapping problem is that, most of the time, the models and their mappings are hard-coded in their supporting tools. As a consequence, they have limited flexibility for modifications and customizations [4].

In order to solve these problems, this work introduces the Transformation Profile concept. The Transformation Profile is intended to externalize the knowledge of how to transform the AUI model to the CUI model. Fig. 1 illustrates the use of a Transformation Profile. 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 to the CUI model. To organize the transformation knowledge, the Transformation Profile is structured in a set of Model Mappings and a Transformation Template. In other words, **one Transformation Profile = one set of Model Mappings + one Transformation Template**.

The Transformation Profile approach provides flexibility for the modification and customization of transformation rules, as well as interesting reusability potential.

Model Mappings and Transformation Templates are introduced in the following subsections.

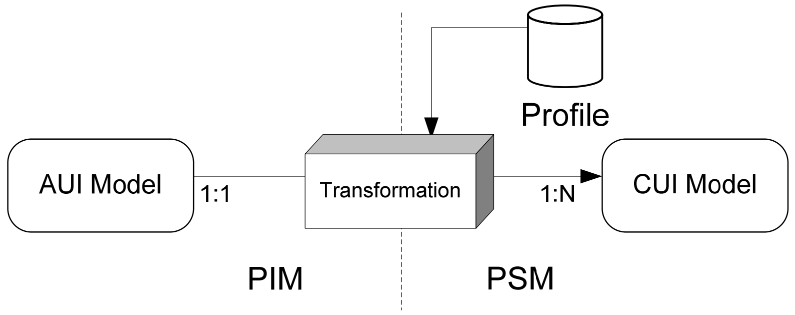


Fig. 1. AUI Model to CUI Model transformation using a Transformation Profile

2.1 Model Mappings

A mapping model is a well-known issue in the MDA of UIs. Puerta and Eisenstein [12] presented a general framework to solve the mapping problem in model-based UI development systems. Following the same line, Montero et al. [7] introduced a formal definition of potential mappings among UsiXML models with its corresponding syntax. UsiXML is a XML-compliant UI Description Language (UIDL) that allows designers to apply a multi-directional development of UIs at multiple levels of independence (http://www.usixml.org). In the UsiXML Mapping Model, the *isReifiedBy* relationship indicates that a CIO is the reification of an AIO through a transformation (see [7] for more details). This relationship has been used in this work in order to define the model mappings.

Our Mapping Model is composed of relationships of reification type between an AUI model and a CUI model. Each mapping is specified by one Source, zero or more Conditions, one Target, and a Priority. The Source is an AIO of the AUI model, the Target is a CIO of the CUI model. A graphical CUI model can represent a UI in terms of CIOs that can be containers (such as *window, horizontalBox*, etc) or individual components (such as *outputText*, *inputText*, etc.) [15]. The containers can contain other containers or individual components defining a tree-like structure. Therefore, the Target could be a CIO which is the root of a CIO tree. If Conditions are specified, each of them must be satisfied in order to the Source be reified in the Target. A Condition is a Boolean expression that can be specified over elements of the AUI model. Finally, the Priority allows specifying the prevalence of some mappings over others. The Conditions and Priority constitutes extensions over the UsiXML Mapping Model.

To clarify the idea of the mappings, a simple textual example is given: let *input* be an AIO that represents the input argument of a method of the domain model, let *horizontalBox* be a concrete container, and let *outputText* and *inputText* be concrete individual components; the Mapping Model allows us to specify the reification of the *input* into an *horizontalBox* that contains an *outputText* at the left and an *inputText* at the right.

The Model Mappings allow the designer to specify widget selection and layout. In addition, different Model Mappings can be defined to address different UI platforms and end-user preferences.

2.2 Transformation Templates

In order to give more flexibility to the transformation from the AUI model to the CUI model, a Transformation Template is used in conjunction with the set of Model Mappings. A Transformation Template is composed of parameters that specify how the CUI model and subsequent final UI are going to be structured and/or stylized.

A model is composed of elements that have attributes with their corresponding data types and values. Well-defined meta-models specify default values for the attributes of their elements. The Transformation Template parameterizes the model transformation with parameters that overwrite default values of attributes of CIOs of a CUI model, e.g. style parameters like colours or font types.

High-level parameters, which are not directly related to a single attribute of an element, can also be specified in a Transformation Template. These parameters can be related to a group of attributes of one or more elements, or to the elements themselves and relations among them. Several customizations can be achieved with high-level parameters gathered in the Transformation Template. There can be parameters for specifying the widgets to be used, the layout options, the dialog style (e.g., by *wizard* or by *tabbed dialog box*), the location of objects (e.g., position of a *toolbar*) or the alignment of elements (e.g. alignment of *labels* with respect to their associated input elements). Furthermore, high-level parameters can overwrite some of the mappings of the previously defined Model Mapping.

Each parameter is described by its name, set of possible values, default value, and the elements where it is applied.

The scopes of application of the parameters can be specified in two dimensions: UI fragments or UI patterns.

For the UI fragment dimension, the following scopes of application specify that the parameter is applied to:

- *Intra-application*: all fragments of the application UI.
- *Inter-container*: all UI containers of a particular type (e.g., *windows*, *dialog boxes*, *tabbed dialog boxes*, *toolbars*) within the application.
- *Intra-container*: all UI containers of a particular type with all their contained UI fragments (e.g., *images*, *icons*, *widgets*) within the application.
- *Inter-individual-component*: all UI individual components of a particular type (e.g., all *buttons*).

Some of the above categories can be combined to obtain more refined applications. For instance, by combining the inter-individual-component and inter-container scopes, the parameter will be applied only to a particular type of UI individual component within a particular type of UI container (e.g., *buttons* of a *dialog box*).

For the UI pattern dimension, the following scopes of application specify that the parameter is applied to:

- *Inter-pattern*: all UI patterns.
- *Intra-pattern*: a UI pattern of a specific type.
- *Inter-sub-pattern*: all UI sub-patterns of a specific UI pattern.
- *Intra-sub-pattern*: a specific UI sub-pattern of a particular UI pattern.

Once developed, a Transformation Template can be applied to a range of interactive applications, for instance, in order to ensure compliance with corporate style guides or to make a family of applications consistent in their look and feel. Besides, some of the parameters could be implemented in a user preference´s configuration file in the final software product, so as to enable the final users to adapt some aspects of the UI to their personal preferences (colours, font types, position of windows, etc.) by means of a suitable editor. The adherence to style guides and the adaptability affect the usability of a software product [3].

The use of high-level parameters and the combinations of their scopes of application give a lot of flexibility and power to the notion of Transformation Profile.

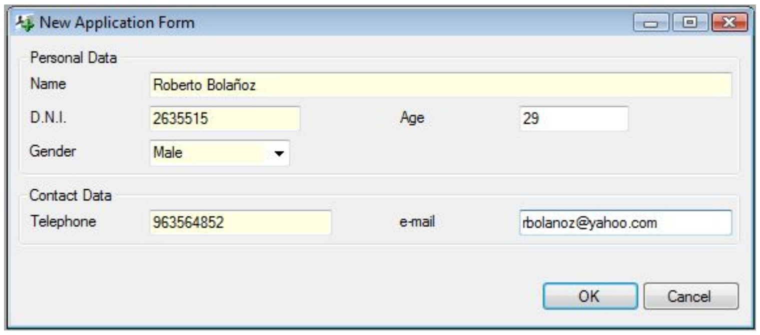
3 Applying Transformations Profiles in the Generation Process of User Interfaces in OO-Method: A Case Study

OO-Method [10] is a software development method that is MDA-compliant. It uses models in order to specify the structural and functional aspects of information systems. It also uses a Presentation Model (PM) [6] that is based on interface patterns in order to specify the UI in an abstract way. OO-Method is supported by OlivaNova - The Programming Machine (a commercial product of CARE Technologies – http://www.care-t.com/) that edits the various models involved and automatically applies subsequent transformations until the final code of a fully functional application (not limited to database or UI) is generated.

The first level of the OO-Method PM is made up of Interaction Units (IUs) that represent the main interactive operations to be performed. One of the IUs is the *Service IU* which is used for specifying the presentation of a service that modifies an object, their attributes and relationships. The next level of decomposition of the PM consists of restricting and specifying the behaviour of each IU using elementary patterns. In a *Service IU* the following elementary patterns, among others, could be defined:

- *Argument Grouping*: enables the arrangement of input arguments of a service in groups and subgroups, and establishes the order in which groups and input arguments are shown to the user. An *Argument Grouping* element of type group corresponds to a group of input arguments, while an *Argument Grouping* element of type argument corresponds to one input argument.
- *Defined Selection*: enables the definition of a set of valid values and can be associated to an input argument.
- *Introduction*: allows the specification of edit masks, valid value ranges, and help and validation messages, and can be associated to an input argument.

The application selected to illustrate the Transformation Profile approach is a photography agency management system. Consider a *Service IU*, of the OO-Method PM, to register photographers. In the registration process, the photographers must supply personal data: name, D.N.I., age, gender; and contact data: telephone and e-mail. The *Service IU* is structured with two *Argument Grouping* elements of type group (Personal Data and Contact Data), which contains *Argument Grouping* elements of type argument which are related to the input arguments of the service (name, D.N.I., telephone, etc.). A *Defined Selection* pattern is used to specify a set of valid values for gender: male and female; and an *Introduction* pattern is used to define a valid value range for age (between 0 and 120). Fig. 2 reproduces the described *Service UI* as generated by OlivaNova for a desktop platform.


New Application Form

Personal Data

Name	Roberto Bolañoz		
D.N.I.	2635515	Age	29
Gender	Male		

Contact Data

Telephone	963564852	e-mail	rbolanoz@yahoo.com

Fig. 2. UI generated by OlivaNova for the photographer registration example

It is important to note that the OO-Method PM corresponds to an abstract representation of a UI without any details of the visual appearance. The OlivaNova transformation engine generates the source code of the UIs from this model by applying transformation rules that are implicit in the tool. Therefore, if the final UI does not satisfy the end-user´s requirements, manual modifications must be applied.

Pederiva *et al.* [11] introduced a Beautification Process for OO-Method in order to address the shortcomings related to the generation of the UIs and manual modifications. The first step of the process proposes to derive a CUI model from the OO-Method PM. In the mentioned work, the UsiXML CUI model was selected for this purpose.

A UsiXML CUI model consists of an abstraction of a final UI independently of the particular widgets used in a specific computing platform, thus resulting in a characterization of a UI in terms of CIOs. In this work, only graphical CIOs, such as *separator* (a decorator), *inputText* (a graphical individual component), or *window* (a graphical container) are considered. Further details about the CIOs provided by UsiXML can be found at http://www.usixml.org/documentation/usixml1.8.0/UsiXML.xsd.html.

The Transformation Profile approach could be introduced into the OO-Method UI generation process to add flexibility to the PM to UsiXML CUI model transformation. Fig. 3 illustrates the proposed evolution for the OO-Method UI generation process.

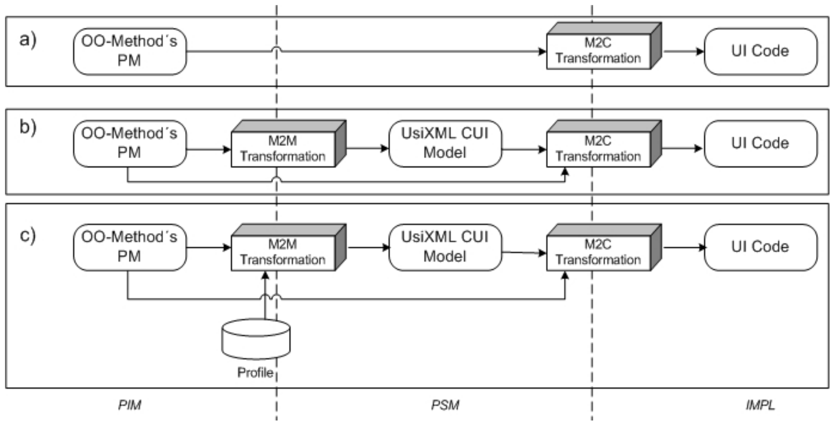


Fig. 3. OO-Method UI generation process: a) in its current state; b) as proposed in [11]; c) using a Transformation Profile

Table 1 represents a subset of the Mapping Model that externalizes the mappings used by the OlivaNova compiler in the generation of the UI shown in Fig. 2. The Source column represents an interface pattern from the OO-Method PM, the Conditions column lists the conditions that must be satisfied for the mapping to be applied, and the Target column shows the UsiXML CUI Model transformation result.

Table 1 presents the mappings in ascendant order of priority, so that, for example, an *ArgumentGrouping* of type argument, related to an input argument of type integer or string, which has a *Defined Selection* associated, will be mapped to a *comboBox*.

Table 1. Mapping Model (subset) for PM to UsiXML CUI Model transformation

Source	Conditions	Target
Service IU		*window* that contains a *borderBox* which encloses a *topBox* and a *bottomBox*. The *topBox* contains a vertical-oriented *box*. The *bottomBox* contains a right-aligned *flowBox* with OK and Cancel *buttons*
Argument Grouping	*Argument Grouping* type is group	*groupBox*
	Argument Grouping type is argument and *Argument Grouping* is related to a string or integer input argument	horizontal-oriented *box* that contains an *outputText* and an *inputText*
Defined Selection		*comboBox*
Introduction		*inputText*

The default mappings provided by OlivaNova are enough from a functional point of view but do not always meet the customer's requirements. To solve this problem, an alternate Transformation Profile could be applied. Table 2 represents a subset of an alternate Mapping Model that allows *Service IUs* with more than one *Argument Grouping* element of type group to be displayed like a wizard. This option can be useful when several input arguments must be entered and the user need only focus on one arguments group. Furthermore, the *Defined Selection* pattern is mapped to a *radioButton,* and the *Introduction* pattern of type integer that specifies a valid values range is mapped to a *spin*. Table 2 presents the mappings in ascendant order of priority.

In order to provide a better customized UI, the alternate Transformation Profile includes a Transformation Template, which is represented in Table 3. The Transformation Template defines font properties (*textFont* and *isItalic*) and a vertical alignment of the labels for all the UI fragments of the application. The *labelAlignment* is a high-level parameter that overwrites the mapping number 4 of Table 2. Furthermore, visual and font properties are specified for all the containers of type *window*, and all the individual components of type *button*.

Table 2. Alternate Mapping Model (subset) for PM to UsiXML CUI Model transformation

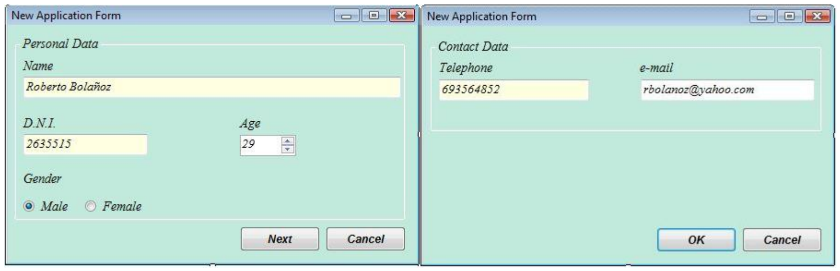
Source	Conditions	Target	Mapping Number
Service IU	*Service IU* does not have *Argument Grouping* elements of type group	*window* that contains a *borderBox* which encloses a *topBox* and a *bottomBox*. The *topBox* contains a *groupBox*. The *bottomBox* contains a right-aligned *flowBox* with OK and Cancel *buttons.*	1
Argument Grouping	*Argument Grouping* type is group and the *Argument Grouping* is the last group of a *Service IU*	*window* that contains a *borderBox* which encloses a *topBox* and a *bottomBox*. The *topBox* contains a *groupBox*. The *bottomBox* contains a right-aligned *flowBox* with OK and Cancel *buttons*	2
	Argument Grouping type is group and the *Argument Grouping* is not the last group of a *Service IU*	*window* that contains a *borderBox* which encloses a *topBox* and a *bottomBox*. The *topBox* contains a *groupBox*. The *bottomBox* contains a right-aligned *flowBox* with Next and Cancel *buttons*.	3
	Argument Grouping type is argument and *Argument Grouping* is related to a string or integer input argument	horizontal-oriented *box* that contains an *outputText* and an *inputText*	4
Defined Selection		*radioButton*	5
Introduction	*Introduction* type is integer and *Introduction* specifies a valid value range	*spin*	6

Table 3. Transformation Template (subset) for PM to UsiXML CUI Model transformation

Parameter name	Parameter value	Scope of application (UI fragment)	Container to apply	Individual component to apply
textFont	Times New Roman	intra-application	all	all
isItalic	yes	intra-application	all	all
labelAlignment	vertical	intra-application	all	all
bgColor	C2EADD	inter-container	all *windows*	none
isBold	Yes	inter-individual-component	none	all *buttons*
textFont	Arial	inter-individual-component	none	all *buttons*

The Transformation Profile, which is composed of the Mapping Model and the Transformation Template presented in Tables 2 and 3, could be an input for the

Model Compiler so as to generate a UsiXML CUI model from which the final UI could be generated according to the required changes. This approach enables the designer to choose the most suitable Transformation Profile for a concrete UI development. In addition, a Transformation Profile repository can be created to reuse previously defined UI specifications. Fig. 4 represents the UI which could be obtained if the new Transformation Profile is applied.



Personal Data
Name: Roberto Bolañoz
D.N.I.: 2635515
Age: 29
Gender: [✓ Male] Female

Contact Data
Telephone: 693564852
e-mail: rbolanoz@yahoo.com

Fig. 4. Expected UI applying the alternate Transformation Profile

4 Related Work

The approach described in this paper is original since it combines a set of Model Mappings and a Transformation Template in a single Transformation Profile to support transformations that are tailored to each application.

Some software tools support a transformation-based approach for generating a UI (e.g., Teallach [2], TERESA [8]), but the transformations are not made explicit and, therefore, they cannot be edited or parameterized. In particular, MOBI-D [12] and Mastermind [14] cannot be considered as genuine transformation approaches since only the models are explicit: the transformations are not explicit and there is no true transformation engine. Mastermind is based on a rule-based approach while MOBI-D directly generates code from the models. TransformiXML [5] does support explicit transformations since it interprets mappings written in UsiXML and converts them into graph transformations. Although these transformations are explicit, and therefore can be edited, they cannot be conditioned, prioritized nor parameterized, which limits their flexibility. In TransformiXML, the designers themselves must enter new transformation options, which is a complex process reserved to specialists, as opposed to parameterizing existing transformations thanks to their parameters. This tailoring process is much more affordable to designers.

There are also other software tools that support a template-based approach, but they are restricted to only modifying the values of some widget properties. For instance, Genova (http://www.esito.no/) gathers predefined values of UI properties, like colour, font, and style, in a template that is then applied to a UI. Our approach generalizes the notion of template to high-level parameters and also handles the notion of UI pattern, which, as far as we know, does not exist in similar works.

When comparing the Transformation Template with Cascading Style Sheets (CSS) [16], we can say that while CSS is a mechanism for adding style to Web documents, the Transformation Template is thought to be used in a MDE-compliant UI development life cycle in order to specify, not only the style, but also the structure of UIs for different computing platforms (desktop, web, mobile). Furthermore, parameters of the Transformation Template can be associated to UI patterns besides UI containers or individual components.

To the best of our knowledge, no existing work today provides both a transformation-based approach (e.g., based on Model Mappings) and a template-based approach (e.g., based on Transformation Templates) in a single and unified way of developing UIs. This combination enables us to combine the powerfulness of the first approach with the flexibility of the second.

5 Conclusion

To summarize, the contribution of this paper is twofold:

1. From the *conceptual* viewpoint, it has introduced the notion of Transformation Profile, which consists of a Transformation Template and a set of Model Mappings to be applied during the model-to-model transformation steps in MDE of UIs. The Transformation Profile externalizes the transformation rules and makes them editable, customizable and reusable. The Model Mappings can be conditioned and prioritized. With regard to the Transformation Template, the different application dimensions (UI fragments and UI patterns) allow designers to apply the parameters in the same way as the *selector* does in CSS. Since the Transformation Profile is independent of the underlying models, nothing prevents its reuse in any other work in MDE of UIs. In principle, the Transformation Profile notion could be used in any model-to-model transformation or model-to-code compilation.
2. From the *methodological* viewpoint, this approach has been incorporated in OO-Method, which is a MDE method for automatically generating an entire interactive system (and not just the UI).

Nowadays, the hardest challenge consists of identifying the parts of the tools to be expanded when new parameters need to be incorporated. A new abstraction could be included in the model editor, but this would affect the high-level modelling activity and may introduce unnecessary levels of detail at this step. A new parameter could be inserted in the various transformation steps, but this would largely affect the transformation engine implementation. Therefore, we think that the easiest solution is to introduce a Profile *during* the transformations themselves. Of course, this still affects the model-to-code compiler, but only in a way that augments its capabilities in an incremental way.

Therefore, the most important shortcoming of this approach relies in its implementation cost. Even though this cost is relatively high this approach allows designers to apply the Transformation Profile to tailor the MDE process to end-users needs. End-users love to specify their own needs and really appreciate seeing them incorporated in the MDE process, as opposed to a traditional MDE process where all the transformations are predefined and leads to a predetermined UI.

References

1. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15,3 (2003) 289–308
2. Griffiths, T., Barclay, P., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C., and Pinheiro da Silva, P.: Teallach: a Model-based User Interface Development Environment for Object Databases. Interacting with Computers 14, 1 (2001) 31–68
3. ISO/IEC 9126-1 (2001) Software engineering - Product quality - Part 1: Quality model
4. Limbourg, Q., Vanderdonckt, J., Addressing the Mapping Problem in User Interface Design with UsiXML. In: Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA2004 (Prague, November 15-16, 2004), Palanque P, Slavick P, Winckler M (eds.). ACM Press, New York (2004) pp. 155–163
5. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In: Proc. of 9th IFIP Engineering Human Interaction and Interactive Systems EHCI-DSVIS’2004 (Hamburg, July 11-13, 2004). Lecture Notes in Comp. Science, Vol. 3425, Springer, Berlin (2005) pp. 200-220
6. Molina, P.J., Meliá, S., Pastor, O.: JUST-UI: A User Interface Specification Model. In: Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI2002 (Valenciennes, May 2002). Kluwer Academic Press, Dordrecht (2002) pp. 63–74
7. Montero, F., López-Jaquero, V., Vanderdonckt, J., González, P., Lozano, M. and Limbourg, Q. Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. In: Proc. of 12th Int. Workshop on Design, Specification and Verification of Interactive Systems DSV-IS2005 (Newcastle upon Tyne, 13-15 July 2005), Harrison M (ed.). Lecture Notes in Computer Science, Vol. 3941. Springer, Berlin (2005) pp. 161–172
8. Mori, G., Paternó, F., Santoro, C.: Design and Development of Multi-device User Interfaces through Multiple Logical Descriptions. IEEE Trans. on Soft. Engineering 28(8) 507–520
9. Object Management Group, MDA Guide Version 1.0.1, 2003. http://www.omg.org/docs/omg/03-06-01.pdf. Accessed 25 January 2008
10. Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice. A Software Production Environment Based on Conceptual Modeling. Springer, New York (2007)
11. Pederiva, I., Vanderdonckt, J., España, S., Panach, I., Pastor, O.: The Beautification Process in Model-Driven Engineering of User Interfaces. In: Proc. of 11th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT2007 (Rio de Janeiro, September 10-14, 2007). Lecture Notes in Computer Science, Vol. 4662. Springer, Berlin (2005) pp. 409–422
12. Puerta, A.R., Eisenstein, J.: Towards a General Computational Framework for Model-Based Interface Development Systems. Knowledge-based Systems 12 (1999) 433–442
13. Schmidt, D.C.: Model-Driven Engineering. IEEE Computer 39, 2 (2006) 41–47
14. Szekely, P., Sukaviriya, N., Castells, P., Muthukumarasamy, J., Salcher, E.: Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach. In: Proc. of 6th IFIP Int. Conf. on Engineering of Human-Computer Interaction EHCI’95 (Yellowstone, August 1995). Chapman & Hall, London (1996) pp. 120–150
15. Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005). Lecture Notes in Computer Science, Vol. 3520. Springer, Berlin (2005) pp. 16–31
16. World Wide Web Consortium (2007) Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. http://www.w3.org/TR/CSS21/. Accessed 28 April 2008.