Transformation Templates: Adding Flexibility to Model-Driven Engineering of User Interfaces

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

[1]Centro de Investigación en Métodos de Producción de Software, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain

[2]Université catholique de Louvain, Louvain School of Management (LSM), Place des Doyens, 1 - B-1348, Louvain-la-Neuve, Belgium

{naquino, opastor}@pros.upv.es, jean.vanderdonckt@uclouvain.be

ABSTRACT

Model-based user interface (UI) development environments are aimed at generating one or many UIs from one or many 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 paper introduces "Transformation Templates", an approach that is adding flexibility to the MDE of UIs by externalizing the transformation logic of UI models, and making it editable, customizable, and reusable. It is also intended to make it easier for UI designers to specify the transformations. A Transformation Template specifies a series of parameters that enable designers to parameterize the model transformation process at the concept level that is of a higher level of abstraction than at the level of physical properties of UI widgets. This paper presents an editor for Transformation Templates and an example of Parameter Type. Transformation Templates can be effectively and efficiently used in any circumstances where the transformation knowledge needs to be modified by non-experts, such as in domain specific languages where flexibility is 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), User Centered Design.* I.3.6 [**Methodology and Techniques**]: Device Independence, Language, Standards.

General Terms

Design, Human Factors, Standardization, Languages.

Keywords

Model-driven engineering of user interfaces, user interface model, user interface model transformation, template, parameter.

1. INTRODUCTION

Approaches of Model-Driven Engineering (MDE) of User Interfaces (UIs) aim at providing an environment where UI designers and developers can design and implement UIs in a professional and systematic way, more easily than when using traditional UI development tools, such as toolkits and Integrated Development Environments made for programming languages, but not for UIs [7]. Model-Driven Engineering of User Interfaces (MDE-UIs) implies the definition of UI models of high abstraction from which UI models of lower abstraction are obtained successively through model-to-model (M2M) transformations until the source code of the UI is reached through a model-to-code (M2C) transformation. M2M and M2C transformations are done automatically or semi-automatically. These approaches introduce well-defined and automatable processes in which good practices of UI generation can be embedded.

Although MDE-UIs incorporates some advantages in the UI development process, at the same time, it creates a new problem: how could UI designers intervene in the automatic MDE process when they need to develop UIs with characteristics that go beyond the scope of the model-driven process they are using? [7]. First of all, why would UI designers need to contribute to the automatic MDE process they use? Basically, they will need to intervene if the MDE process is not powerful enough to generate the kind of UIs they need. One identified challenge in MDE of UIs is the "*need for powerful transformation and rendering engines*" [21]. 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. Second, why would UI designers need to develop UIs with characteristics that go beyond the scope of an automatic MDE process, even if they use a powerful MDE process? This need 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*" [21].

Having justified the need of UI designers to intervene in the MDE of UIs, the next paragraphs analyze how designers intervene in current approaches for MDE of UIs. On the one hand, there are approaches for MDE of UIs which have their transformation logic and design knowledge implicit and fixed in the tools that perform the transformations (e.g., Teallach [9], TERESA [15], OLIVANOVA [16]). 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 ([17, 11]), since for that purpose, they must manually tweak the generated UI code. On the other hand, there also exist approaches for MDE of UIs which have their transformation logic explicitly expressed, either with mapping and transformation models or with model transformation languages (e.g., TransformiXML [12]). In these cases, the customization can be made prior to the UI generation. However, UI designers must define the new transformation options that they need, or edit existing ones. That complex process is more oriented to model transformation specialists than to UI designers.

In general, end-user requirements related to the UIs make it necessary that UI designers intervene in the MDE of UIs. However, taking into account the described situation of MDE of UIs, 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 the model of a UI to 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 [5].

In order to face the described problem, in a previous work [2] we introduced Transformation Profiles as a mechanism to externalize and customize the UI model transformations and to reuse knowledge between different UI development projects. A Transformation Profile is composed of a set of Model Mappings (see [2] for more details) and a Transformation Template. This work reviews and enhances the specification of Transformation Templates. A Transformation Template is composed of Parameters that specify details of the structure, layout and style of UIs. They parameterize UI model transformations.

Four different abstraction levels with their corresponding models are usually used to structure the UI development life cycle: Task and Concepts, Abstract User Interface (AUI), Concrete User Interface (CUI) and Final User Interface (FUI) [5]. Following a forward engineering process, the previous abstraction levels give rise to three types of transformations in the MDE of UIs: (1) from the Task and Concepts Model to the AUI Model; (2) from the AUI Model to the CUI Model; and (3) from the CUI Model to the FUI code. The Transformation Templates approach can be applied on these three types of transformations. Furthermore, it can also handle transformations from UI models on patterns [14, 20].

The rest of the paper is organized as follows: Section 2 presents the Transformation Template approach. Transformation Templates and their associated concepts are described with meta-models. Furthermore, an example of parameter type is presented and the automation of Transformation Templates is briefly mentioned. In Section 3, related works are presented. Finally, Section 4 presents some conclusions.

2. TRANSFORMATION TEMPLATES

A Transformation Template has the aim to specify the structure, layout and style of a UI according to preferences and requirements of end-users, and the different hardware and software computing platforms and environments in which the UI will be used.

A Transformation Template is composed of Parameters with associated Values which parameterize the UI model transformations. Note that the term "Transformation Template" has been used to denote transformation rules in XML transformation languages. IBM has also used the term to denote template models with associated UML profiles. However, in this work, the term is related to parameterized transformation processes. This externalizes the transformation logic and makes it customizable according to the characteristics of the project which is being carried out. Transformation Templates can then be reused in other projects with similar characteristics. Figure 1 illustrates the use of a Transformation Template when transforming from an AUI model to a CUI model. A transformation engine takes as input an AUI model and a Transformation Template. The Transformation Template provides specifications which determine how to transform the AUI to the CUI model. The transformation engine follows the specifications to generate the CUI model.

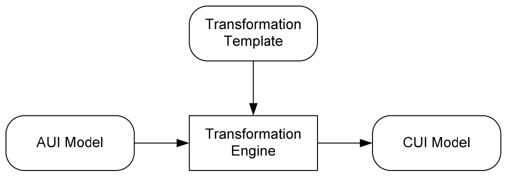


Figure 1. Transformation from an AUI Model to a CUI Model using a Transformation Template.

The concepts that characterize the Transformation Templates approach belong to one of three groups: concepts related to the Contexts of use of the UIs, to UI Models, and to Transformation Templates themselves. The groups of Contexts and UI Models provide concepts which support the specific Transformation Template concepts. In order to precisely describe and characterize all concepts related to Transformation Templates, meta-models have been elaborated for each one of the groups. The next section defines the meta-models for each group of concepts.

2.1 Context of use

In this work the context concept refers to the context of use of an interactive system. The purpose of conceptualizing context is that we want it to be possible to define different Transformation Templates for different contexts of use. We have adopted a view of the UsiXML Context Meta-Model in order to characterize the context of use of interactive applications[1]. Figure 2 illustrates the elements and relationships which make up the considered view.

A *Context of use* is composed of the stereotype of a user who carries out an interactive task with a specific computing platform in a given surrounding environment. A *User Stereotype* represents a set of end-users who share the same values in some descriptive parameters or properties: language, experience doing a task, experience using the system, among others. The *Environment* describes any property of interest of the physical environment where the end-user interacts with the system. Such properties may be physical (e.g., lighting conditions), psychological (e.g., level of stress), or organizational (e.g., location and role definition in the organization chart). The *Platform* captures relevant attributes for each combined software-hardware platform and attached devices that may significantly influence the context of use in which the end-user carries out an interactive task. A platform may consist of a series of physical hardware devices, a series of software components, the characteristics of the network to which the platform is connected, the capability to support wireless, and the capability of browsing web pages.

[1] The complete UsiXML Context Meta-Model can be found in http://www.usixml.org/index.php?mod=pages&id=5

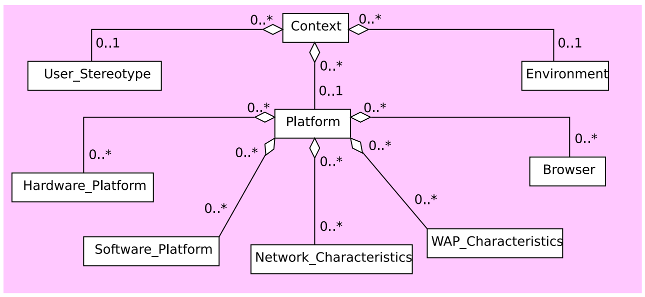


Figure 2. Elements and relationships of the considered view of the UsiXMLContext Meta-Model.

The considered view of the UsiXML Context Meta-Model gives support to the context concept as defined by the Cameleon Reference Framework (CRF) [5] which is widely accepted in the Human-Computer Interaction community.

2.2 UI Models

This work presents a Transformation Templates approach which is used to parameterize transformations among UI models. Therefore, Transformation Templates should be able to be used with different existing approaches for UI modeling, such as [1, 12, 15].

Therefore, we need to specify how to relate Transformation Templates with different UI modeling approaches. In order to do this, we have defined a meta-model of UIs which represent, in a simplified way, some of the concepts that characterize different meta-models of UIs. This simplified version provides only the necessary and sufficient concepts in order to explain how Transformation Templates are defined and how they should be used. At the same time, they define the requirements that a UI modeling approach must fulfill in order to be able to be used with Transformation Templates. The Transformation Templates approach assumes that UI Meta Models are composed of Meta Elements and Meta Relationships among Meta Elements (Figure 3).

A *UI Meta Model* represents, in a general and simplified way, different existing meta-models of UIs (abstract, concrete or pattern based meta-models).

A *Meta Element* is an element of a UI Meta Model. It is an abstract element whose specializations are Meta Container and Meta Individual Component. At an abstract, concrete or final level, as well as in pattern based approaches, a UI can be characterized as a hierarchy of containers holding individual components. Individual components are atomic and cannot be further decomposed, while containers can contain individual components or other containers. A Meta Individual Component represents an individual component and a Meta Container represents a container.

A *Meta Relationship* is composed of a set of Meta Elements which are the source of the relationship and a set of Meta Elements which are the target. The Meta Relationship is an abstract element whose specialization is the Meta Decomposition. A Meta Decomposition represents the logical structure of components of a UI. In a Meta Decomposition the relationship with source Meta Elements is specialized to a relationship in which all Meta Elements must be Meta Containers. The Meta Elements which are the target of a Meta Decomposition can be Meta Individual Components or Meta Containers.

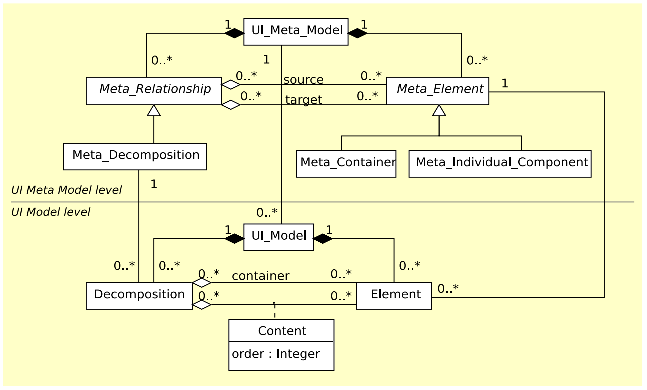


Figure 3. Elements and relationships of the UI meta-model considered in this work.

All previously mentioned UI concepts are located in the UI Meta Model level in Figure 3 since they are necessary to represent different existing UI Meta Models.

In the UI Model level, which represents different existing UI Models (abstract, concrete or pattern based models), the *UI Models* are composed of Elements and Decompositions. A UI Model is associated to its corresponding UI Meta Model (see Figure 3).

An *Element* represents an element of a UI Model and is associated to its corresponding Meta Element which can be a Meta Individual Component or a Meta Container.

A *Decomposition* has a set of Elements which are containers in the Decomposition as well as a set of Elements which are contained. The contained Elements are ordered. A Decomposition is associated to its corresponding Meta Decomposition and this association determines which Elements can be containers or be contained in the Decomposition.

2.3 Transformation Templates

Figure 4 illustrates the elements and relationships which make up the part of the Transformation Template meta-model which is used to define Parameter Types, this is to say, the Parameter Type definition level. It also illustrates the relationships of such elements with elements of the UI Meta Model level, Context meta-model, and the Parameter definition level of the Transformation Template meta-model which will be explained latter.

A *Parameter Type* specifies the characteristics that will be shared by a set of Parameters. These parameters establish how a UI will be structured and stylized. A Parameter Type is an abstract element and its specializations are Presentation and Dialog. These elements are, in turn, further specialized (see Figure 4). The specializations of Parameter Type allow defining parameters which affect different aspects of a UI. Font, Colour, Layout and Navigation are low-level Parameter Types. This means they operate at attribute level of UI models specifying values for attributes such as font type or colour. High-Level Presentation and High-Level Dialog are high-level Parameter Types and they operate at the concept level of UI models specifying, for instance, the widgets to be used, the location and alignment of objects.

Defining a Parameter Type subsumes specifying the Meta Elements that are affected by it. Saying that a Parameter Type affects a set of Meta Elements means that the corresponding Parameters will be able to have an effect (e.g., a visual effect in the UIs) on the Elements related to the Meta Elements in question.

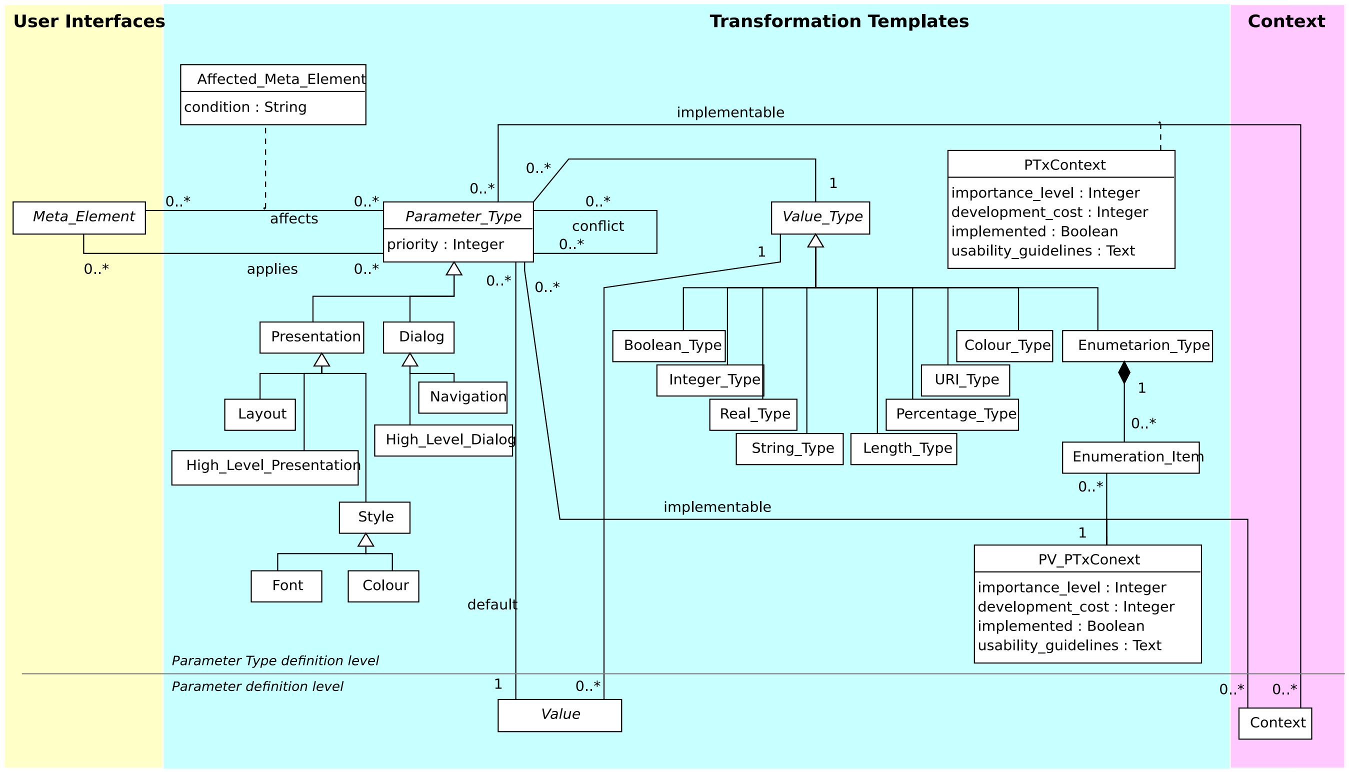


Figure 4. Elements and relationships of the Parameter Type definition level of the Transformation Template meta-model.

This is descriptive information and can be useful for the person who will implement the Parameter Type in a M2M or M2C compiler. It is also possible to specify a condition in order for Parameters associated to the Parameter Type to affect the corresponding Elements. The condition must be a Boolean expression which can involve Meta Elements and Parameter Types.

The definition of a Parameter Type also involves the specification of the Meta Elements on which the Parameter Type can be applied. In this case, saying that a Parameter Type applies to a set of Meta Elements means that the corresponding Parameters will be able to be applied to the Elements related to the Meta Elements in question. The Meta Elements on which a Parameter Type can be applied are those which are affected by it as well as other Meta Elements which can contain the affected ones, in any contention level. A Parameter Type *PT* which affects the Meta Element *ME1* could be applied to *ME1* and/or to another Meta Element *ME2*, provided *ME2* can contain *ME1* in any contention level. So, if we consider a Parameter *P* of type *PT*, an Element *E1* related to *ME1*, and an Element *E2* related to *ME2* which contains *E1*, when applying *P* to *E1* its effect will be reflected directly on *E1*, and when applying *P* to *E2* its effect will be reflected, also, in *E1*.

A Parameter Type could be in conflict with other Parameter Types. The conflict can occur when the Parameter Types affect the same Meta Element causing the same effect (e.g., two or more Parameter Types have an effect in the font type to be used in the Elements which correspond to a same Meta Element). In order to partially resolve conflicts, each Parameter Type has a priority. Conflicts are resolved giving preference to the effect of the Parameter Type with higher priority. In case the priorities of the conflicting Parameter Types are equal, the conflict must be resolved in a random way or according to a criterion determined by who implements the Transformation Templates approach in a M2M or M2C compiler.

A Parameter Type has an associated *Value Type*. The Value Type concept is similar to the notion of data type or type of object (class). It is an abstract element and its current available specializations are Boolean Type, Integer Type, Real Type, String Type, Length Type, Percentage Type, URI Type, Colour Type and Enumeration Type. The Enumeration Type has an associated set of Enumeration Items. The specialization set includes types commonly used in UI related specifications (e.g., in [4]) and could be enlarged in the future. A Parameter Type also has a default value which must be associated to its Value Type.

When a Parameter Type has a Value Type which is an Enumeration Type the corresponding Enumeration Items constitute its possible values. Each of these possible values is implementable or not in different Contexts (see the relationship among Parameter Type, Enumeration Item and Context in Figure 4). From this set of implementable possible values it must be decided which ones will indeed be implemented in the given Context. This is an important decision since the implementation cost could be high as it implies modifications in M2M and/or M2C compilers. Each possible value of a Parameter Type receives estimations of its importance level (IL) and its development cost (DC). The IL estimation should be based on the importance of the possible value for the Context in consideration, as well as in the frequency with which end-users ask for changes in UIs which can be related to the Parameter Type being analyzed. The DC should be estimated by programmers who have experience programming for the Context in question. Possible values with high IL and low DC will have the higher priority when deciding which possible values to implement for a given Context.

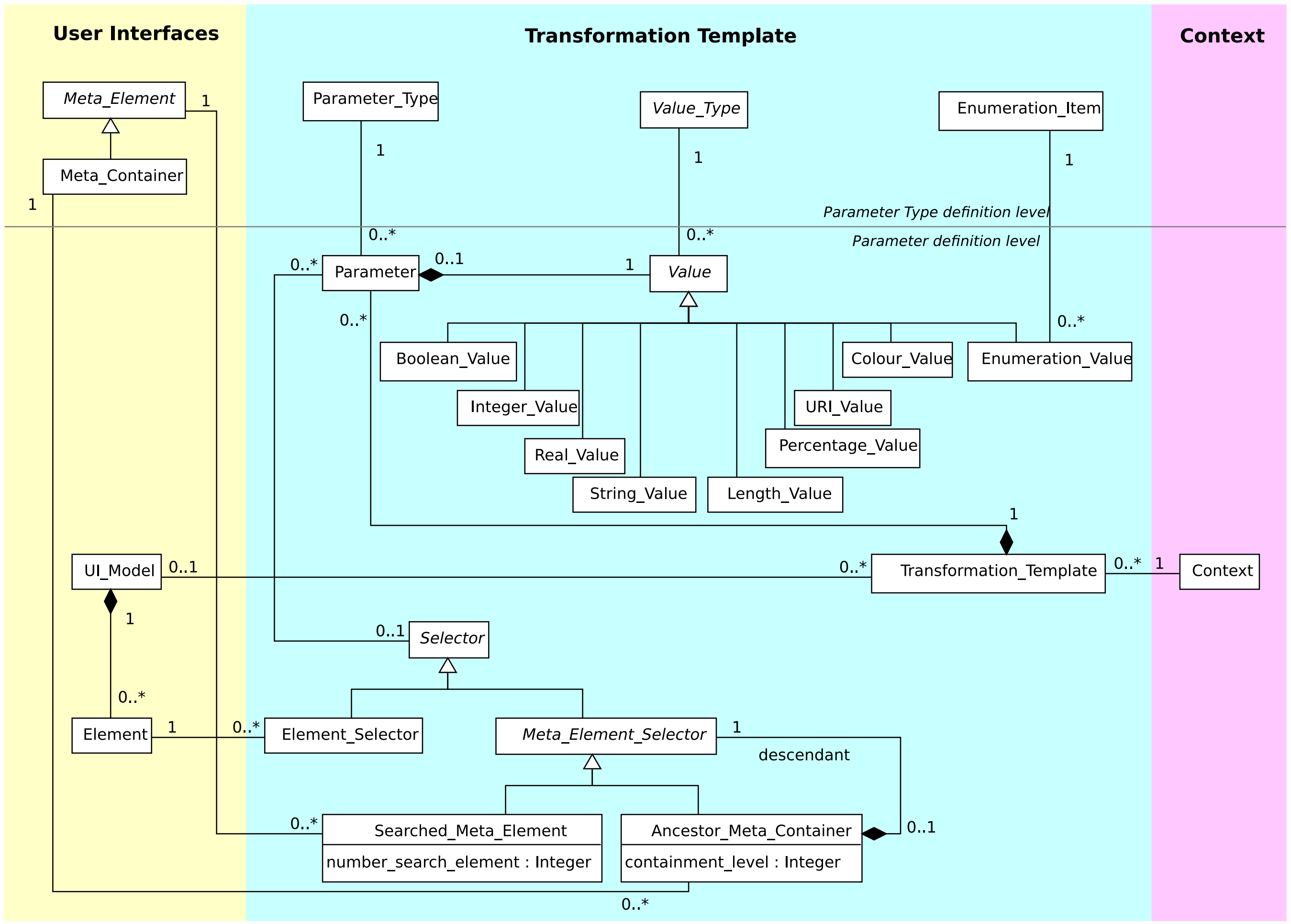


Figure 5. Elements and relationships of the Parameter definition level of the Transformation Template meta-model.

The second level of priority is characterized by high IL and DC; the third level, by low IL and DC; and finally, the fourth level, by low priority and high DC. Furthermore, each possible value of a Parameter Type can be assigned with usability guidelines which will help UI designers in choosing one of the possible values explaining in which conditions they are convenient or not to use.

In a similar way, a Parameter Type is implementable or not in different Contexts (see the relationship between Parameter Type and Context in Figure 4). From the set of implementable Parameter Types it must be decided which ones will indeed be implemented in a given Context. In the case of a Parameter Type which has a Value Type that is not an Enumeration Type, the process for deciding if it will be implemented in a given Context and which will be its implementation priority, is exactly the same as the one previously explained for the case of possible values of Parameter Types. In the case of a Parameter Type which has a Value Type which is an Enumeration Type, its data related to the Context is derived from the data of its possible values in the same Context. That is to say, it will be implementable in a given Context if one of its possible values is implementable in that Context. Its IL and DC estimations will be aggregations of the IL and DC estimations of its possible values, respectively. And it will be considered implemented in a given Context if one of its possible values is implemented in that Context.

Figure 5 illustrates the elements and relationships which make up the part of the Transformation Template meta-model which is used to define Transformation Templates and Parameters, this is to say, the Parameter definition level. It illustrates the relationships of such elements with elements of the UI Meta Model level, UI Model level, Context meta-model, and the Parameter Type definition level of the Transformation Template meta-model.

A *Transformation Template* gathers a set of *Parameters* for a specific Context of use and can be associated, or not, to a UI Model. Each Parameter of a Transformation Template corresponds to a Parameter Type, has a Value and can have a Selector.

A *Value* corresponds to a Value Type. The Value concept is similar to the notion of data or object. It is an abstract element and its current available specializations are Boolean Value, Integer Value, Real Value, String Value, Length Value, Percentage Value, URI Value, Colour Value and Enumeration Value. Each of these values corresponds to a Value Type. The Value which is assigned to a Parameter must be related to the Value Type associated to the Parameter Type which corresponds to the Parameter in question. When a Parameter is assigned with a Value which is an Enumeration value, its associated Enumeration Item must be implemented for the Context of the Transformation Template in which the Parameter is located. If the Value is not an Enumeration Value it will be enough with the Parameter Type related to the Parameter being implemented for the Context of the Transformation Template.

A *Selector* has the aim to delimit the set of Elements of a UI Model over which a Parameter is applied. This, in turn, delimits the set of Elements which are affected by the Value of the Parameter. If a Parameter does not have an associated Selector, the Parameter will be applied to all Elements associated to Meta Elements over which the Parameter Type corresponding to the Parameter in question is able to be applied. If the Parameter has an associated Selector, it will only be applied to the Elements selected by the Selector. Selector is an abstract element and its specializations are Element Selector and Meta Element Selector.

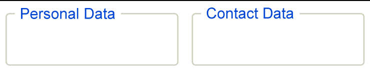
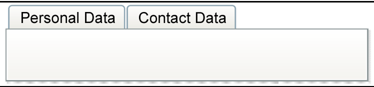
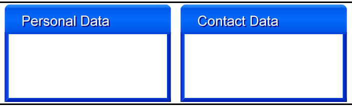
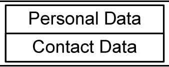
An Element Selector allows the selection of a specific Element from a UI Model. The Parameter associated to this Selector will be applied on the selected Element. It is necessary that the selected Element corresponds to a Meta Element over which the Parameter Type of the Parameter is able to be applied. In order to associate Element Selectors to the Parameters of a Transformation Template, the Transformation Template must be associated to the UI Model from which Elements will be selected.

A Meta Element Selector is an abstract element and its specializations are Searched Meta Element and Ancestor Meta Container. A Searched Meta Element selector allows the selection of Elements of a specific Meta Element. If the specified Meta Element is a Meta Container, the number of the searched element can also be specified. In this case, the Searched Meta Element selector selects the Element contained at the position specified by the number of the searched element. The position is given by the order of the Element in the Decomposition. We have defined special options for the number of the searched element. The special options allow the selection of all Elements but the first one, the last Element, and all Elements but the last one. A Parameter associated to a Searched Meta Element selector will be applied on all selected Elements. It is necessary that the selected Elements correspond to Meta Elements over which the Parameter Type of the Parameter is able to be applied. Furthermore, a Searched Meta Element selector can have associated an Ancestor Meta Container with a containment level. In this case, it is also necessary for an Element to be contained, at the level specified by the containment level, in other Element related to the Ancestor Meta Container, in order for the first Element to be selected by the Searched Meta Element selector. An Ancestor Meta Container can, in turn, have another Ancestor Meta Container with containment level, and this can be repeated until the required level.

2.4 How to use the Transformation Templates Approach

There are two main processes related to the Transformation Templates approach: the definition of Parameter Types, and the creation of Transformation Templates. The definition of Parameter Types begins with the identification of a required UI customization by end-users of automatically generated UIs or by UI designers.

Table 1. Grouping Layout Parameter Type.

Parameter Type			
		Possible values enumeration	
Name	***Affects***	***Value***	***Graphical description***
Grouping layout for input arguments	Two patterns of the OO-Method Presentation Model: Service Interaction Unit and Argument Grouping	group box	
		tabbed dialog box	
		wizard	
		accordion	

	Contexts			
	SW: C# on .NET - HW: laptop or PC		*SW: iPhone OS - HW: iPhone*	
Possible value	***Importance level (IL)***	***Development cost (DC)***	***Importance level (IL)***	***Development cost (DC)***
group box	high	low	high	low
tabbed dialog box	high	low	medium	medium
wizard	medium	medium	low	high
accordion	low	medium	medium	medium

Possible value	***Usability guidelines (for desktop context)***
group box	Visual distinctiveness is important. The total number of groups will be small.
tabbed dialog box	Visual distinctiveness is important. The total number of groups is not greater than 10.
wizard	The total number of groups is between 3 and 10. The complexity of the task is significant. The task implies several critical decisions. The cost of errors is high. The task must be done infrequently. The user lacks the experience it takes to complete the task efficiently.
accordion	Visual distinctiveness is important. The total number of groups is not greater than 10.

Then, a UI designer creates the corresponding Parameter Type and, maybe with the help of a MDE expert, specifies the affected Meta Elements from one or many UI Meta Models. The UI designer also has to specify the possible values for the Parameter Type, as well as the contexts in which those values are significant. Then, for each context, the UI designer has to estimate the IL of each possible value based on experience and feedback from end-users. Also, for each context, an experienced programmer has to estimate the DC of each possible value, and a usability expert has to incorporate usability guidelines. The decision of implementing or not a Parameter Type for a given context and in a given MDE approach should be based on the corresponding IL and DC estimations. If the decision is positive, a programmer must proceed with the implementation.

Regarding the creation of Transformation Templates, the first step consists in defining the context for which the Transformation Template will be created. Then, a specific UI Model can optionally be associated to the Transformation Template, and finally, Parameters can be added to the Transformation Template. When adding a Parameter, its value and selector must be specified.

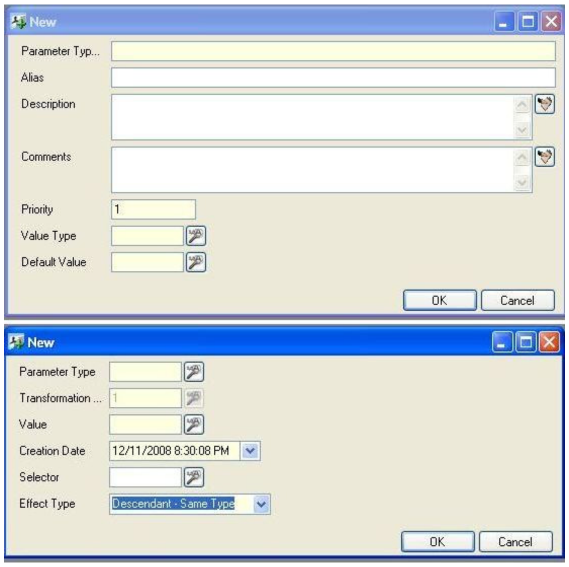
2.5 An Example of Parameter Type

Table 1 presents the Grouping Layout Parameter Type. This Parameter Type was extracted from a Catalogue of Parameter Types which was proposed for the OO-Method Presentation Model (PM). OO-Method [16] is an object-oriented method which allows the automatic generation of software applications from conceptual models. The OO-Method PM is structured with a set of UI Patterns which are defined in [14] and allow the specification of an abstract representation of a UI. OLIVANOVA *The Programming Machine* is a commercial tool developed by CARE Technologies (http://www.care-t.com) which supports OO-Method. Currently, most of the logic that guides UI model transformations is implicit in the OLIVANOVA compiler so we thought the application of the Transformation Templates approach could be convenient.

The Service Interaction Unit and Argument Grouping that appear in the affects column of Table 1, are elements of the OO-Method PM (see [14] for more details about them). Four different possible values have been defined for the Parameter Type, and they have been associated to two context of use: a desktop platform and a mobile one. For each context of use and each possible value the IL and DC have been estimated. Furthermore, for the desktop context and each possible value, usability guidelines have been proposed from an extraction from [8].

2.6 Transformation Templates Automation

The automation of the Transformation Templates approach implies two types of tools: Transformation Template editors, and M2M/M2C compilers which use Transformation Templates in its compilation process. By now, we have implemented a first prototype of a Transformation Template editor. It has been developed using OO-Method and the OLIVANOVA technology. Figure 6 presents two screenshots of the editor. The top window allows creating a new Parameter Type. Other UIs of the editor allow the user to complete all the steps which are required to complete the specification of the Parameter Type: association with the affected Meta Elements and with significant contexts of use, estimation of IL and DC, and specification of usability guidelines. The bottom window allows creating a Parameter in a Transformation Template. As future work, we want to improve this editor including visualization facilities when creating Transformation Templates.


New

Parameter Typ...
Alias
Description
Comments
Priority 1
Value Type
Default Value

New

Parameter Type
Transformation ... 1
Value
Creation Date 12/11/2008 8:30:08 PM
Selector
Effect Type Descendant - Same Type

Figure 6. Screenshots of the Transformation Templates editor.

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 [9], TERESA [15], OLIVANOVA [16]). With respect to these works, this approach has the advantage that logic transformation and design knowledge are externalized and, therefore, are customizable and reusable in UI development projects with similar characteristics. Other software tools support explicit transformations. TransformiXML [12] interprets mappings written in UsiXML [12] and converts them into graph transformations. Furthermore, different model transformation languages had been applied in MDE of UIs (e.g., ATL [10], RDL/TT [19], XSLT [6]). A survey of some of them is presented in [18]. With respect to these works, this approach has the advantage that it incorporates a parameterization process which is more affordable for designers.

There are also other software tools that support a template based approach (e.g., Genova [3], CSS [4]). With respect to these works, this approach has the advantage that it does not depend on a particular UI development method but it can be used with different approaches for MDE of UIs. 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 parameters can be added when needed. This approach is also applicable to model transformations for patterns.

In [13], another type of Transformation Templates, used for performing transformations among XML documents, is automatically generated. In our case, the purpose is not to automatically generate our templates because in the UI design context, the design knowledge, held in the experience of UI designers, needs to be turned into Transformation Templates in order to maximize the quality of the results. In fact, creating a Transformation Template could be a high-cost activity for some contexts of use. However, the high-cost will be relative if the Transformation Template allows the generation of good-quality UIs and if it is reused among UI development projects.

4. CONCLUSION

The contribution of this work is related to the definition of the Transformation Templates approach which intends to add flexibility in UI model transformations. The Templates allow the specification of details of the structure, layout and style of UIs. They are inputs for M2M or M2C compilers and parameterize the logic which compilers follow to perform the transformations. In this way, Transformation Templates externalize the design knowledge and make it customizable according to the characteristics of a UI development project. They also allow the proper logic to be reused in other projects with similar characteristics. Moreover, this parameterization process is simple compared to the complex process of specifying model transformations with mapping and transformation models or model transformation languages.

The concepts related to the Transformation Template approach have been clearly characterized. The approach has also been related to a simplified and generic UI meta-model. The establishment of these relationships allows Transformation Templates to be used with different approaches of MDE of UIs. In a similar way, the establishment of relationships with a Context meta-model allows Transformation Templates to guide transformations for different end-users, hardware and software platforms, and environments. Furthermore, in principle, the Transformation Template notion could be used in any M2M or M2C compilation.

The set of Parameter Types is not fixed. Different approaches of MDE of UIs could define their own Parameter Types, many of which could be useful in different approaches for MDE of UIs. In order to avoid a tedious process for the specification of details of a UI, the Transformation Templates approach provides different flexible mechanisms: default values are specified for Parameter Types and several ways of selecting the elements which will be affected by a Parameter are provided.

The proposed approach allows relating usability guidelines to Parameter Types in order to guide the UI designer towards a Transformation Template which can produce UIs with good usability. The implementation cost can be identified as the most important shortcoming of this approach since the implementation of each Parameter Type implies modifications in compiler tools. However, these modifications can be done in an incremental way. Even though this cost is relatively high this approach allows designers to apply the Transformation Template to tailor the MDE process to end-users needs. End-users desire 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 lead to a predetermined UI.

5. ACKNOWLEDGMENTS

This work has been developed with the support of MEC under the project SESAMO (TIN2007-62894), by GVA under grant BFPI/2008/209, and the ITEA2 Call3 UsiXML project under reference 2008026.

6. REFERENCES

[1] ABRAMS, M., PHANOURIOU, C., BATONGBACAL, A. L., WILLIAMS, S. M., AND SHUSTER, J. E. UIML: An Appliance-Independent XML User Interface Language. *Computer Networks 31*, 11-16 (1999), 1695–1708.

[2] AQUINO, N., VANDERDONCKT, J., VALVERDE, F., AND PASTOR, S. Using Profiles to Support Model Transformations in the Model-Driven Development of User Interfaces. In *Computer-Aided Design of User Interfaces VI, Proc. of 7th Int. Conf. on Computer-Aided Design of User Interfaces, CADUI 2008, (Albacete, Spain, June 11-13, 2008)* (2009), Springer, Berlin, 2009, pp. 35–46.

[3] ARISHOLM, E., BENESTAD, H. C., SKANDSEN, J., AND FREDHALL, H. Incorporating Rapid User Interface Prototyping in Object-Oriented Analysis and Design with Genova. In *In Proceedings of NWPER'98 Nordic Workshop on Programming Environment Research (Eds, Mughal* (1998), pp. 155–161.

[4] BOS, B., ÇELIK, T., LIE, H. W., AND HICKSON, I. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Tech. rep., World Wide Web Consortium (W3C), July 2007.

[5] 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.

[6] CLARK, J. XSL Transformations (XSLT) Version 1.0. W3C recommendation, W3C, Nov. 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[7] DA SILVA, P. P. User Interface Declarative Models and Development Environments: A Survey. In *DSV-IS* (2000), pp. 207–226.

[8] GALITZ, W. O. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[9] GRIFFITHS, T., BARCLAY, P. J., PATON, N. W., MCKIRDY, J., KENNEDY, J. B., GRAY, P. D., COOPER, R., GOBLE, C. A., AND DA SILVA, P. P. Teallach: a Model-Based User Interface Development Environment for Object Databases. *Inter. with Comp. 14*, 1 (2001), 31–68.

[10] JOUAULT, F., AND KURTEV, I. Transforming Models with ATL. In *MoDELS Satellite Events* (2005), J.-M. Bruel, Ed., vol. 3844 of *Lecture Notes in Computer Science*, Springer, pp. 128–138.

[11] LIMBOURG, Q., AND VANDERDONCKT, J. Addressing the Mapping Problem in User Interface Design with UsiXML. In *TAMODIA* (2004), P. Slavk and P. A. Palanque, Eds., ACM, pp. 155–163.

[12] LIMBOURG, Q., VANDERDONCKT, J., MICHOTTE, B., BOUILLON, L., AND LÓPEZ-JAQUERO, V. USIXML: A Language Supporting Multi-path Development of User Interfaces. In *EHCI/DS-VIS* (2004), vol. 3425 of *Lecture Notes in Comp. Sci.*, Springer, pp. 200–220.

[13] LLAVADOR, M., AND CANÓS, J. H. A Framework for the Generation of Transformation Templates. In *ECDL* (2007), vol. 4675 of *Lecture Notes in Computer Science*, Springer, pp. 501–504.

[14] MOLINA, P. J., MELIÁ, S., AND PASTOR, O. Just-UI : A User Interface Specification Model. In *CADUI* (2002), C. Kolski and J. Vanderdonckt, Eds., Kluwer, Dordrecht, 2002, pp. 63–74.

[15] MORI, G., PATERNO, F., AND SANTORO, C. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Trans. Software Eng. 30*, 8 (2004), 507–520.

[16] PASTOR, O., AND MOLINA, J. C. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Springer-Verlag, 2007.

[17] PUERTA, A. R., AND EISENSTEIN, J. Towards a General Computational Framework for Model-Based Interface Development Systems. *Knowl.-Based Syst. 12*, 8 (1999), 433–442.

[18] SCHAEFER, R. A Survey on Transformation Tools for Model Based User Interface Development. In *HCI (1)* (2007), vol. 4550 of *Lecture Notes in Computer Science*, Springer, pp. 1178–1187.

[19] SCHAEFER, R., DANGBERG, A., AND MUELLER, W. RDL/TT - A Description Language for the Profile-Dependent Transcoding of XML Documents. In *In International ITEA Workshop on Virtual Home Environments* (2002).

[20] VALVERDE, F., PANACH, J. I., AQUINO, N., AND PASTOR, Ó. *New Trends on Human-Computer Interaction. Research, Development, New Tools and Methods*. Springer, 2009, ch. Dealing with Abstract Interaction Modelling in an MDE Development Process: a Pattern-based Approach, pp. 119–128.

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