Designing User Interface Adaptation Rules with T:XML

Víctor López-Jaquero, Francisco Montero, Fernando Real
Laboratory of User Interfaces and Software Engineering (LoUISE)
Instituto de Investigación en Informática (I3A)
University of Castilla-La Mancha, 02071 – Albacete (SPAIN)
{ victor | fmontero | freal }@dsi.uclm.es

ABSTRACT

The specification of model adaptation and generation rules is a topic of great interest for the user interface development community, since there are more and more approaches supporting the model-based approach. The ubiquitousness in interaction and the different user profiles are not the only challenges when designing interactive systems. Furthermore, the context of use evolves over time. In this situation, there is a strong need to provide a set of adaptation rules to make the user interface evolve according to the context of use evolution. This paper contributes a metamodel for the definition of adaptation rules in a systematic approach, pursuing engineer adaptation. Moreover, a tool called T:XML is presented that supports the specification of adaptation rules using a visual notation that greatly simplifies the process of designing adaptation for model-based user interface environments.

Author Keywords
user interface development environment, user interface adaptation, T:XML tool.

ACM Classification Keywords
D.2.2 **[Software Engineering]:** Design Tools and Techniques – *user interfaces*. H.5.2 **[Information Interfaces and Presentation]:** User Interfaces*, Graphical User Interfaces (GUI)*.

INTRODUCTION

With the advent of ubiquitousness in interaction and the quick advances in technology many different devices are appearing. Moreover, the contexts of use where those devices are being used transcend the desktop to reach the streets or many other physical environments.

On the other hand, currently the profile of the users of software applications is shifting more and more towards a less advanced user profile. The traditional advanced user profile, usually even with some programming skills, has become a small percentage among the mass of users.

This situation, where many factors are levering how interaction should take place, poses interesting challenges to user interface (UI) developers. Many users are requiring applications that can be run in different platforms and environments. Users require applications supporting making the same tasks with different devices and in different places. However, creating applications covering this requirement is complex and expensive, since in the worst possible scenario, a different UI should be created for each supported context of use (CoU).

To overcome this problem, model-based user interface development environments [15] are used in UI development. They are aimed at providing a systematic approach to specify the UI by means of models. Later, these models will be translated into the final code executed by the user. Nevertheless, a set of rules is required to generate the final code for the UI adapted to every target CoU supported.

In most cases, the final code for every target CoU cannot be generated a priori during design stages, since the CoU itself evolves over time. One example of this kind of environment is an e-learning system where the user (the knowledge he/she has) evolves as learning is carried out. The most common approach to solve this problem is to provide a set of adaptation rules. These adaptation rules will provide the artifacts to make the UI evolve according to the user's evolution.

Thus, adaptation rules should be a must in today's UI development practices. Nevertheless, in most of the systems providing adaptation, these rules are design in ad-hoc manner [3]. Therefore, most of these systems are using what we call *hardcoded adaptation* techniques. In this hardcoded adaptation the adaptation code is mixed with the rest of the code of the application, making reusing the adaptation capabilities designed almost impossible. The path towards raising the level of abstraction in adaptation design leads to what we call *engineered adaptation* [6].

In this paper a metamodel for the specification of general purpose and reusable adaptation rules is presented. This metamodel contributes to help in engineering adaptation. Moreover, a tool that supports the design of the UI adaptation produced by those rules is described also. Next, a review of some related work is discussed.

DESIGNING USER INTERFACE ADAPTATION

As aforementioned, in the model-based approach to UI design a set of models are used to specify the system. These models are transformed from the more abstract models to the more concrete ones. Nevertheless, a set of rules to specify how the models are transformed is required.

Many different approaches can be found in the literature regarding the transformation of models into other models or code. These different approaches are aimed at the development of general systems, but some of them are specific (or have been adapted) for their use in the development of UIs.

According to [2] the following generic approaches for model transformation are being used: (1) *direct-manipulation approaches*. (2) *structure driven approach*. (3) *relational approaches*. (4) *graph transformation based approaches*. (5) *template based approach*. (6) *operational approach*. (7) *hybrid approach*. (8) *other approaches*: there are other approaches, such as XSLT [15], that doesn't fall in any of the previous categories, but they are also commonly used in model transformation.

On the other hand, there are some approaches in UI model transformation frameworks that adapt some of the aforementioned techniques. For instance, in [5], an adaptation of the graph transformation approach for the development of UIs can be found. In this approach, the UI model expressed in terms of UsiXML[14] is transformed by a set of task using the attributed graph grammars. For this purpose, AGG tool [13] API is used. According to [9] there are a set of desirable features a model transformation approach should cover:

- *Transformation rules*: the approach should provide a language supporting the specification of transformation/adaptation rules.
- *Rule application control*: a mechanism to control what rules to apply is required, since not every rule is applicable in a given context.
- *Rule organization*: reusing the transformations is a key feature. By reusing transformations we are not just reusing the time spent in creating the rules, but also the adaptation design experience gathered during the development.
- *Source-target relationship*: it is the ability of the transformation approach to be able to generate several output models.
- *Incrementally*: because changes can occur both due to changes in the requirements and changes in the CoU, it is necessary to provide the ability to update an existing model.
- *Directionality and tracing*: ideally, the transformations should be bidirectional, and log of what transformations have been applied should be maintained. This feature supports a very important issue in adaptation: undoing adaptations.

These desirable features are applicable to both UI generation by model transformation and UI adaptation. Currently, the only approach covering all these features is QVT [10]. Nevertheless, the way QVT transformations are specified is complex, and it requires additional knowledge about OCL (Object Constraint Language).

We have been using the graph transformation approach for some years now [5]. Although, the way the transformations are specified if quite visual when using AGG tool [13], the amount of resources required for the transformation of an average UI is somehow slow for its use in real-time environments.

On the other hand, XSLT is a very popular approach among software developers, and there are plenty of tools supporting both the design and the execution of XSLT. It provides a versatile tool to express the adaptation rules. However, *rule organization*, *rule application control* and *incrementally* must be implemented by the developer. Furthermore, it does not support *directionality and tracing*.

Our approach to model transformation for adaptation

Because of our good experience in the design of adaptation rules following the graph transformation approach, we have worked in the design of a transformation approach covering all our requirements. One requirement not usually considered in model transformation approaches is portability. Portability is paramount to UI adaptation, since it must work for the different platforms the software was designed, from desktop PCs to PDAs. This requirement can be overcome by using client-server architecture for adaptation, but it requires a network connection constantly, that can eventually exhaust quickly the battery in portable devices.

Thus, in our approach we are using a graph transformation approach to the design of transformations, but these transformations are internally converted by our tool to generate XSLT code. This code can be executed in almost every platform by using either Java-based implementations or Javascript-based implementations.

Nevertheless, by using this approach we are not supporting *directionality and tracing*. To support *directionality* we have implemented an undo layer above XSLT. This undo layer takes advantage of the information regarding the applied rules stored in the model transformed. This kind of information is stored by means of mappings (or links) [8] that express the relationships between the elements in the models being transformed and the elements created in the target models. These mappings are supported by user interface description languages (UIDL) such as UsiXML [14], XIML or the Teallach approach.

Source-target relationship requirement for model transformation can be implemented in XSLT by using

XSLT 1.1 or 2.0. In those versions of XSLT, several output files can be produced by a single transformation.

The definition of an adaption rule is not just the specification of the transformation it produces in the models, but also the context where it is applicable and the events that trigger the adaptation rule. Unfortunately, there is no approach in UI design to specify how adaptation rules should be described. Therefore, reusing the adaptation rules from one application to another is a tedious work, because the rules must be converted from one metamodel to another. Next, our metamodel for the definition of adaptation rules is described.

A METAMODEL FOR UI ADAPTATION RULES

In Figure 1 the meta-model for an adaptation rule is depicted. In this meta-model, an adaptativity rule is specified in terms of the context-of-use events that trigger the adaptation rule, the sensors that produce those events, the data the rule accesses (read/write), the transformations of the UI needed in order to apply the rule, and the context precondition.

The adaptation rule is triggered by context events. These events represent the different changes in the CoU detected by the sensors. A context event can be produced by more than one sensor, so that the same rule can be triggered easily by different input events.

The sensors can be either software or hardware sensors. Hardware sensors are built-in in the hardware platform where the application is running. On the other hand, software sensors are pieces of software included in the application to gather some information. For instance, a software sensor could be added to gather the idle time of the user when using our application. Both hardware and software sensors can be specialized in detecting the changes in the platform, the user or the physical environment. One of the most important attributes of sensors is *dataType*. This attributes stores the data type of the piece of information acquired by that sensor.

Each rule has a context precondition. The context precondition specifies the required conditions that the current CoU must meet in order for the adaptation rule to be applicable. For instance, it can express that the value of the parameter acquired by the sensor must be greater than a specific value.

AdaptationRule contains the general information of the rule. It includes a name, a description of the rule descriptive enough to make easier understanding its purpose and a priority. The first level of rule selection is selecting the lowest priority rules. For the specification of the context precondition and the execution of the adaptation the engine must access some models and data. In *data* the required resources for the execution of the rule are specified. This is especially important for distributed or multi-agent systems, where the resources are distributed and shared between different computing entities.

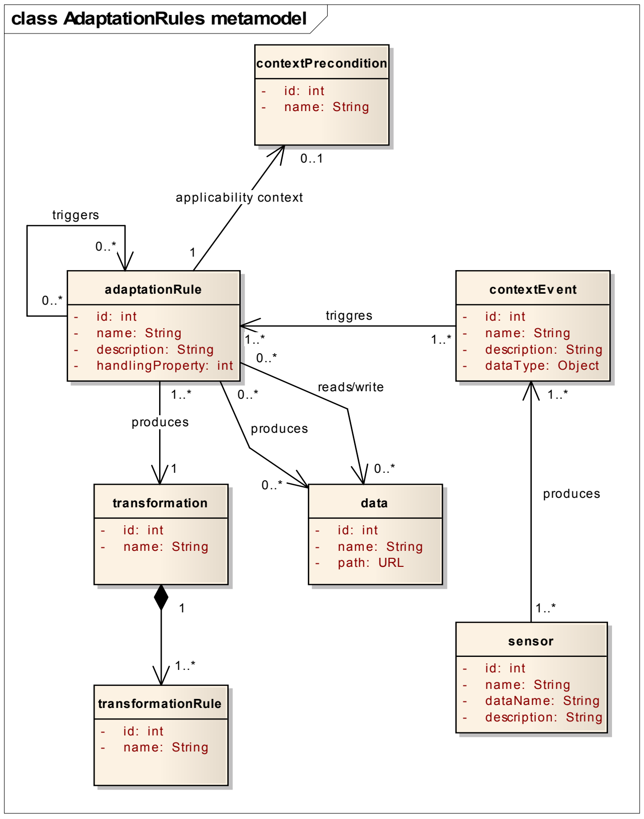


Figure 1: Metamodel for adaptation rules.

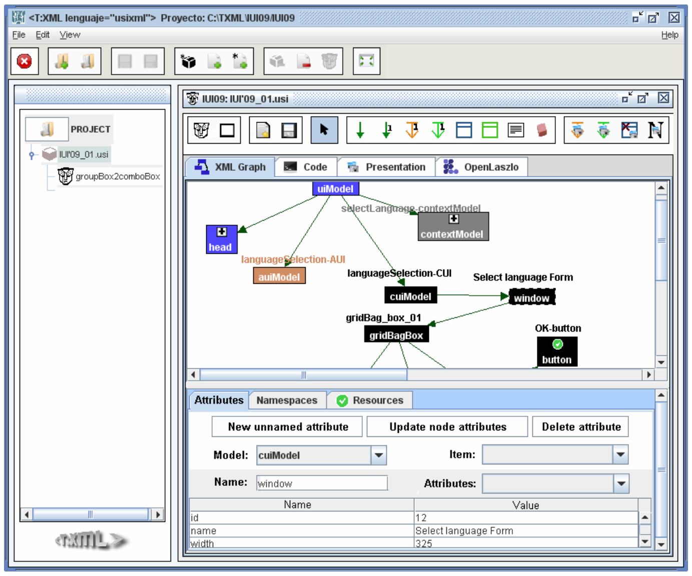
The transformation specifies the "real" adaptation of the UI. It includes a set of model transformations. These transformations can be expressed in any model transformation language. In *transformation* entity the type of the *transformationRules* is stored.

T:XML: A TOOL FOR ADAPTATION RULES DESIGN

T:XML is a tool supporting the specification, organization and validation of adaptation rules. This tool uses an approach for the specification of rules inspired by the graph transformation tool AGG [13]. Currently, the tool supports the definition of adaptation rules that transform models specified in UsiXML language, but support for other UI specification languages such as UIML [4] is planned, since the tool was designed as a generic transformation tool.

The tool organizes the transformations in project folders for an easy management of the rules and the models.

The aim of the tool is the generation of the specified adaptations for different target transformation model languages, such as XSLT or QVT. By generating code for different transforming languages we pursue maximizing the reuse and portability of the designed adaptations. As a proof of concept, the tool currently supports the generation of XSLT code automatically for the visually designed adaptations. Furthermore, the tool supports the generation



Model: cuiModel

Item:

Name: window

Attributes:

Name	Value
id	12
name	Select language Form
width	325

Figure 2: T:XML tool working space.

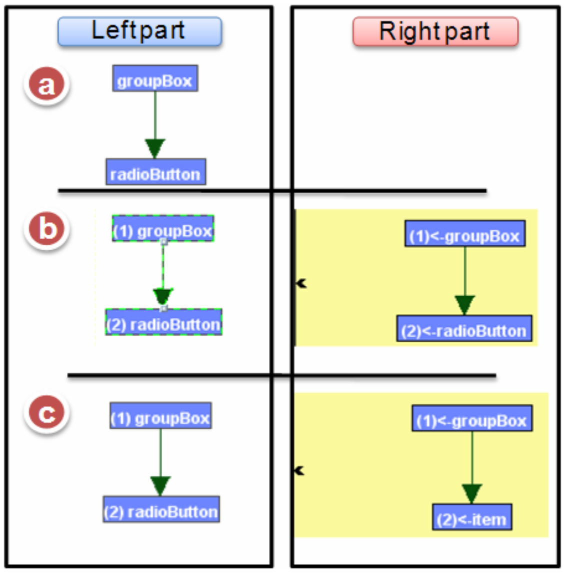


Figure 3: Sample adaptation rule specification.

of the final code of the UI in OpenLaszlo language [11], for previewing and debugging the adaptations designed. By doing so, the designer can easily discover unwanted effects or flaws in the designed adaptations.

Using the tool

There are four zones in the design space of the application: (1) the main toolbar is located at the top of the window. It provides support for file management functions (saving, loading, etc). (2) in the left-most part of the window there is a project manager. This project manager structures the projects and the adaptation rules in folders following a tree layout. In this tree the designer can browse/select the adaptation rules (projects) and the transformation rules. (3) model design space. In this area the designer can visualize imported models or create a new one from scratch. To keep clean this design area, the designer can collapse or expand those branches not currently in use. The branches that have been collapsed are marked with a "+" sign (see *head* element in Figure 2). (4) attribute edition area. In this area the designer can modify the attributes of the elements of the model. Notice some nodes (see *OK-button* element in Figure 2) have a white tick. This tick denotes that that element has an associated value in the resource model. This model is used in UsiXML to provide a separation of the contents from the presentation, supporting internationalization. The resources can be modified in *Resources* tab.

When the user start the transformation rule designer by clicking on the mask icon a transformation design area appears. This area has for parts. It has a tool bar with buttons, and three areas for the specification of the transformation rule. To illustrate how transformation rules are specified a simple example will be used. In our example we want to design an adaptation rule to be executed when the screen space for our application is reduced. Thus, the idea behind our adaptation rule is to replace all the group boxes in our UI containing a set of radio buttons with a *combobox* to reduce the space required to present the selection tasks in our UI.

The central area is the *Left part* specification area. In this area the designer defines the pattern that should be matched in the model. In our example we are searching for *groupBoxes* with some radio buttons in it (see Figure 3a). The right most is the *Right part* specification area. In this area the designer specifies how to transform those nodes matching the left part. Those elements in the left part of the rule not present in the right part will be deleted. In our example we have dragged the left part into the right part see Figure 3b). The matching numbers for *groupBox* and *radioButton* in both parts defines a link between those nodes. This means that the nodes in the left part are kept unchanged, since they appear in the right part. Nevertheless, we want to transform the *groupBox* into a *comboBox* and the *radioButtons* into items for the *comboBox*.

Therefore, in Figure 3c we have change the element types in the right part of the rule to reflect that requirement. Therefore, all the parts of the model to be transformed matching the left part of the rule will be transformed into what we put in the right part. Once the rule has been created is can be saved and applied. The tool automatically creates the target transformation language code. Currently, it generates a set of four XSL transformations. This XSL Transformations can be used in any of the many engines supporting this transformation language. The adaptation can be previewed in our tool in OpenLaszlo framework. The Openlaszlo code is automatically generated by our renderer.

The tool supports also the definition of variables. Variables can be used to apply arithmetic operation to the attributes of the model or to take the value of an attribute to set the value of another attribute. For instance, if in the previous example we create a variable for the width attribute of the *groupBox* we could use this value to customize the value of the width of the *comboBox* created.

DISCUSSION

When creating model transformation rules there is an important issue that should considered. The transformation of the models can produce others models (model-to-model transformations) or code (model-to-code or model-to-text transformations). Model-to-code transformations in UI development are usually used at the final level, when we want to generate the UI that will be presented to the user. Our tool supports model-to-model transformations, covering all the models that can be expressed in a model-based UI development environment (tasks, domain, context, presentation, etc). Nevertheless, the development of a renderer that produces an XML interpretable language, such as XHTML is possible with our tool.

There is another open issue in the development of the adaptation of UIs. When the models are designed they must be finally renderer, so they can be manipulated by the user. We must decide where is the trade-off between the development of the adaptation facilities and the renderer. If the renderer is too simple, all the adaptation required will be designed by the developers of the applications. On the other hand, if the renderer is too complex, it can be hard to customize the adaptation capabilities.

When the designer is creating adaptation rules, is important to note that if the adaptation rules are too detailed they will be hard to reuse in another applications. A good approach would be adding an extra layer to fine control the application of more generic adaptation rules by using a classical rule-based approach, such as CLIPS [1].

CONCLUSIONS AND FUTURE WORK

The specification of model adaptation and generation rules is a topic of great interest for the UI development community, since there are more and more approaches supporting the model-based approach.

In this paper we contribute a metamodel for the specification of adaptation rules. This model allows the specification of the transformations of the UI along with the events that trigger the rule and the sensors that collect the information from the CoU where the application is running.

Furthermore, a tool (T:XML) is presented that supports the specification of these adaptation rules for model-based development environments. The definition of the rules is greatly simplified by providing a graphical tool that hides the complexity behind the creation of the adaptation rules in a programming language. This tool automatically generates the code for the transforming language out of the adaptation rules visually designed by the user.

As future work, to improve the versatility of the tool, we are adding support also for other target transformation languages. We are currently working in the generation of QVT transformation language, since it is one of the most powerful languages supporting bidirectional transformational and tracing of the applied adaptations.

ACKNOWLEDGMENTS

This work is partly supported by the Spanish PAI06-0093-8836 and CICYT TIN2008-06596-C02-01 grants.

REFERENCES

[1] CLIPS. A Tool for Building Expert Systems. http://clipsrules.sourceforge.net/

[2] Czarnecki, K., Helsen, S. Feature-Based Survey of Model Transformation Approaches. IBM Systems Journal, 45(3), 2006, pp. 621-645

[3] Florins, M., Montero, F., Vanderdonckt, J., Michotte, B., Splitting Rules for Graceful Degradation of User Interfaces, Proc. of 10th ACM Int. Conf. on Intelligent User Interfaces IUI'2006, ACM Press, New York, 2006, pp. 264-266.

[4] Helms, J. and Abrams, M. 2008. Retrospective on UI description languages, based on eight years' experience with the User Interface Markup Language (UIML). Int. J. Web Eng. Technol. 4, 2 (May. 2008), 138-162.

[5] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López Jaquero, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. EHCI-DSVIS'2004, Springer, 2005, pp. 200-220.

[6] López Jaquero, V., Montero, F., Molina, J.P., González, P., Fernández-Caballero, A. A Multi-Agent System Architecture for the Adaptation of User Interfaces. CEEMAS 2005. LNAI 3690, Springer, Germany, 2005.

[7] López Jaquero, V., Vanderdonckt, J., Montero, F., González, P. Towards an Extended Model of User Interface Adaptation: the ISATINE framework, EIS 2007, Salamanca, 22-24 March 2007. Springer, 2008..

[8] Montero, F., López Jaquero, V., Vanderdonckt, J., González, P., Lozano, M.D., Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. DSV-IS'2005, Springer, Berlin, Germany, 2005.

[9] Navarro, E. Ph.D Thesis. ATRIUM: Architecture Traced from Requirements by Applying a Unified Methodology, UCLM, 2007.

[10] QVT. MOF Query/Views/Transformations final adopted specification. OMG Document pct/05-11-01, 2005.

[11] OpenLaszlo. Rich Internet Applications framework. http://www.openlaszlo.org

[12] Puerta, A.R. A Model-Based Interface Development Environment. *IEEE Software 14,4* (July/Aug. 1997) 40–47.

[13] Taentzer, G. 2000. AGG: A Tool Environment for Algebraic Graph Transformation. In Proc. of the Int. Workshop on Applications of Graph Transformations with industrial Relevance. LNCS, vol. 1779. Springer, London, 481-488.

[14] UsiXML. USer Interface eXtensible Markup Language. http://www.usixml.org

[15] XSL Transformations (XSLT), version 1.0 W3C recommendation 16/Nov/1999. http://www.w3c.org/TR/xslt