A Survey on Transformation Tools for Model Based User Interface Development

Robbie Schaefer

Paderborn University, C-LAB, Fürstenallee 11,
33102 Paderborn, Germany
robbie@c-lab.de

Abstract. As a wide variety of interaction devices, modalities has to be supported by user interface developers, model-based user interface development gets increasing attention. Especially if context- and user-awareness comes into play, handcrafting a user interface is rendered almost impossible. In model-based user interface development, usually several models are applied to describe different aspects of the user interface or to provide a varying level of detail. The relations between the models representing those levels of abstractions are established through transformations, a concept which is also applied in software engineering with the Model Driven Architecture (MDA). In this paper we will review several transformation systems and discuss their applicability for model-based user interface development.

Keywords: User Interface Engineering, Model Driven Architecture, Model Based User Interface Development, Transformation Tools.

1 Introduction

Model Driven Architecture and Model Based User Interface Development are quite similar since both use models to describe static and dynamic system properties on different levels of abstractions and use transformations from one model to another.

In the Model Driven Architecture (MDA) [19], transformations between different models have been identified to be of key importance [9] and in order to classify the different model transformations, an extensive taxonomy has been proposed by Czarnecki and Helsen [7].

However, classical MDA-approaches have been lacking sound models for the engineering of user interfaces, although some UI development methodologies have been aligned with the OMG standards, as for example the UML based architecture Wisdom [14] and an MDA compliant environment around UsiXML based tools [22].

The Cameleon Reference Framework [5] provides a concensus on the types of UI models used for the different levels of abstractions, namely the tasks and concepts, the abstract user interface (AUI), the concrete user interface (CUI) and the final user interface (FUI). As with the MDA, transformation tools have to be used to move from one layer of abstraction to another or to adapt these models to different contexts of use. As manifold as the number of involved models, their presentations and tools are the used transformation methods in current practice: Some approaches work on the models while other work on their representations, some are integrated in the model while others are applied externally, some are observable and modifiable while others are hardcoded and not accessible.

For this reason, we will review several transformation systems and discuss their applicability for model-based user interface development. The considered transformation approaches and tools are graph transformations (GT) as applied in UsiXML [13], ATL [10], TXL [6], 4DML [4], UIML's internal transformation capability [1], XSLT [11], GAC [8] and RDL/TT [18].

2 Selection and Comparison Criteria

Transformations are an essential part of many domains in computer science and applied computing for example the transformation of programmes, data and models. Since in this survey the focus is on transformations beeing used for engineering of user interfaces, a selection had to be made from the plethora of transformation tools.

Even if we would constrain ourself to model transformation approaches used in the MDA, as done in [7], we would end up in a comparison of more than twenty canditates with a rich set of comparison features. However, model-based user interface development took a variety of paths in the past and provided several other models and transformations than used in software engineering only. For this reason we did not only select model transformation tools from the MDA but also transformation tools which are common practice in model-based UI development or approaches which have interesting properties that may be exploited in engineering user interfaces and in fact have been used for that purpose.

In order to compare the selected transformation tools, we did not go into the same level of detail as in [7] but took a rather practical approach and looked at several criteria which are of particular importance with model-based UI development. First of all, the programming model is compared: While the distinction between imperative and declarative programming does not allow evaluating the expressiveness of the approach it is important with respect to the UI developer's familarity with one or the other approach.

Furthermore, we looked at the capabilities to transform models, XML and code. The distinction between model- and XML-transformation is needed, since many UI-models are described in XML and also a lot of FUIs are XML-based e.g. XHTML, XForms, WML, VoiceXML etc.

Another important aspect is, whether the transformation approach is capable of generating code beyond XML-based FUIs, while the ability of complex mapping as opposed to linear mapping is an evidence for the expressiveness of the approach. Furthermore, we looked at the extensibility and parameterizability of the tools which make the transformation possibilities more versatile, especially with plasticity of the user interfaces for different devices, contexts, users and modalities in mind.

3 Selected Transformation Approaches

Before comparing the selected transformation languages, we will give a brief introduction to each of them and show how they have been used for the transformation of user interface models or representations and as such would with in a model based user interface design modality.

3.1 Graph Transformations (GT) in UsiXML

A formal, purely declarative approach for model transformations is established with graph transformations as shown in [3], since many models can be designed with an underlying structure of directed graph. Graph transformations are quite common for tools in the MDA-domain, for example in AtoM^3 [12]. For this survey, we selected UsiXML as a candidate which used GT since UsiXML is specifically designed for the multi-path development of User Interfaces and one of the first approaches which have been proven to be MDA-compliant [22].

The models UsiXML is based on are based on graphs and therefore the model mappings of UsiXML are specified with graph transformations which consist of a set of transformation rules [13]. Each rule consists of a Left Hand Side (LHS) matching a graph G, a Negative Application Condition (NAC) not matching G and a Right Hand Side which is the result of the transformation. The LHS may also further be augmented by additional attributes to further constrain the matches and thus adding to the expressiveness.

Since graph transformations allow mappings between any models that are based on a graph, UsiXML thus allows reification, abstraction and translation between the models. The limitations with this approach are only with the construction (or reengineering) of the FUI, since the FUI usually is not represented as a graph. Translations between to different FUI formats are also not possible nor intended with UsiXML.

3.2 ATL

Another language used for model transformation is ATL [10]. ATL follows a hybrid approach in a way that the user is in a position to select, whether to use ATL purely declarative or to employ imperative features in addition. The declarative aspect is provided by the approach of matching rules, where a source pattern is described through a set of source types and an OCL-expression which constrains the source types. The target pattern is constructed a similar way by specifying a set of target types form the target meta-model and a set of bindings which are used to initialize the features of the target types.

While this declarative approach is very straightforward, it may be hard to specify more complex rules. For this case, ATL offers to add an action block with imperative constructs to the rules or even allows calling external code for the logic.

Since ATL operates on the models themselves – not even on the representations of the models such as XML-representations – it is not suited as a transcoding tool for other purposes but only for model transformations. ATL has been successfully applied for the model driven engineering of plastic user interfaces [21].

3.3 TXL

TXL [6] is a transformation language which is designed for multiple purposes, especial for the transformation of programming languages, and is not constrained to any source or target format. This is established through two components:

- A specification of the structure to be transformed based on grammars in the Backus Naur Form.
- A set of transformation rules based on pattern/replacement pairs and functional programming

Since the rules are specified in a functional way and the first part of a TXL specification only describes the grammatical structure, TXL can be considered also to be a mostly declarative language. TXL has been also proven to be capable of model transformations [15]. In fact the grammar support allows taking the final step from a concrete model to a representation in a programming language, which is not possible with graph transformations.

3.4 4DML

The transformation language of 4DML (four-dimensional markup language) [4] has been originally developed in order to adapt web content to people with special needs and is therefore considered here. It is designed to transform different notations and as such serves a similar rich application domain as TXL. But while TXL is intended for transforming programming languages which can be represented as a syntax tree, 4DML supports the transformation of data which comes in a matrix-like structure.

The transformation of 4DML documents is done purely declarative through the definition of a source pattern matching and the definition of a target model. While 4DML seems to be strong in transforming between completely different languages where the source is organized in an n-dimensional structure, it is a bit artificial to impose a matrix structure on documents which are organized as trees or graphs.

3.5 UIML Peers

The User Interface Markup Language (UIML) [1] is an XML-based language to describe all relevant aspects of a user interface such as structure, style, content and behavior. A genuine aspect of UIML is the capability to define connections to the backend logic and to provide a vocabulary which maps UIML to other UIML instances or target languages. The latter two aspects are covered in the "peers"-section of UIML and provide the transformation features for this survey. The "presentation"-section of UIML includes mappings of classes (and components) and their properties to target format constructs, while the "logic"-section is used to manage the connection to the application logic.

A presentation section usually has a name, which allows different presentation sections to be provided for different target formats. For example, there may be a presentation section for VoiceXML and for HTML. Mappings of classes and their properties are not necessarily restricted to XMLbased formats but may also be mapped to, e.g., Java constructs.

Since UIML's mapping facility matches class names and provides new values for the matched objects, it can be regarded as declarative. This is however only linear and therefore too simple to support complex restructuring tasks. This results also in limited use, when it comes to model transformation. The obvious advantage of the UIML approach is that the abstract UI-representation and the transformation to the FUI can be specified in the same language. Therefore the model based approach using

UIML as presented in [2] uses UIML internal mappings from the AUI to CUI and from CUI to the FUI but requires an external transcoding approach from the task-level to the AUI.

3.6 XSLT

The transformation language XSLT [11] is designed for the purpose of transforming the XML-based input to textual (mostly XML-based) output. The input of an XSLT program is a set of XML-based documents. The output can be XML, or plain text. With plain text output, an XSLT processor can generate languages different from XML. An XSLT definition defines a set of template rules which associate patterns with templates. Each rule consists of a matching pattern, optional mode and priority attributes, and a template. Matching pattern expressions are defined by a subset of the XPath language and are evaluated with respect to a currently processed (matched) node or the root node. The matching process considers the node's name, attributes, location in the tree and position in the list and results in a set of nodes that can be used to provide parameters for the template or as a base for further matching. XPath supports the processing of node-sets and covers five additional basic types: booleans, numbers, strings, node sets, and tree fragments. Processing usually starts at the root node. When a pattern is successfully matched, the pattern is associated with the template, the template (construction pattern) is recursively executed, mode is possibly changed, and matching is optionally continued from each matched node. For execution, XSLT provides variables and parameters which can be passed between template rules. For pattern processing, XSLT provides literals, constants, variables, and keys (for cross referencing) with conditions, list iterations, recursion, sorting, and numbering as control structures. For advanced processing, XSLT covers a powerful set of built-in string functions for creation, deletion, replacement, copying, and concatenation. While the XSLT processing foundation lies in functional programming, the processing allows imperative statements such as iterations and conditions. Therefore XSLT can be considered to be a hybrid approach.

3.7 GAC

The General Adaptation Component (GAC) [8] has been developed to make web applications more adaptable. In contrast to the other presented transformation languages, GAC provides explicit means to reference context data to control the adaptation process and is able to modify the contextual data. Since its purpose is to adapt web content it is able to process HTML and XML in general. The architecture of GAC is as such notable that it does not use XSLT to describe the transformation rules - which is otherwise a quite common practice in that domain - but provides an RDF-based configuration of the adaptation process.

The GAC configuration consists of rules which are bound to conditions. The rules can be of two types for adaptation and for updating the usage context. The adaptation rules allow deletion and substitution of XML fragments as well separation – the process of sourcing fragments out and making them accessible via links – and the inverse process. As the rules provide clear instructions of which operations to perform when a condition holds, we consider this approach to be more imperative.

3.8 RDL/TT

The Rule Description Language for Tree Transformation (RDL/TT) [18] evolved from a domain specific language for adapting Web-content to different devices into a transcoding language for multiple purposes including context-dependent transformations of XML-based UI descriptions. RDL/TT employs a Java-oriented syntax to define the transformation rules which operate on the DOM-tree of the XML-document. It defines simple search patterns based on tag-names or a collection of tags with complex restructuring rules on the found matches.

A notable property of RDL/TT is the use of variables which may convey contextual information that allow different flows of transcoding operations for varying preferences, target platforms and contexts of use. The transcoding rules are specified in an imperative manner and provide several control structures such as branches and loops together with calls to predefined transcoding functions.

The set of transcoding functions is extensible by compiling additional transcoding libraries to the tool, which for example has been used to include image processing rules to adapt visual content besides the user interface istself. In practice RDL/TT has for example been used for context-based adaptation of web content in [17] and with a generic user interface format in [16]

4 Comparison and Discussion

Table 1 shows the support of different transformation characteristics for the languages we discussed. If a feature is supported, it is marked wit a "+" in the table, if not it is marked with "-". If a supported feature is put in brackets, it means that it is in principle supported (maybe with some additional effort) but that the language is not specifically designed to support that property.

Table 1. Comparison of general transformation language properties

Feature	ATL	GT	TXL	4DML	XSLT	GAC	UIML	RDL
Declarative	+	+	+	+	+	-	+	-
Imperative	+	-	-	-	+	+	-	+
Model Transformation	+	+	(+)	(+)	(+)	(+)	(+)	(+)
XML Transformation	-	-	(+)	(+)	+	+	-	+
Code Transformation	-	-	+	(+)	-	-	-	-
Code Generation	-	-	+	+	(+)	-	+	(+)
Complex Mapping	+	+	+	+	+	+	-	+
Extensible	+	-	-	-	-	-	-	+
Parameterizable	-	-	-	-	-	+	-	+

While the distinction between declarative and imperative programming tells nothing about the capability of the transcoding language, it may be a selection criteria for programmers who feel more familiar in one of these programming models. On the other hand a clear distinction between declarative and imperative transformation systems is not always possible. While for example Graph Transformations (GT) are clearly declarative, the declarative aspects of TXL for instance are a bit diluted. For this reason, the marks for declarative and imperative indicate the strongest tendencies.

With respect to the ability to transform models (and as such to transform UI models), only ATL and graph transformations are really designed for it. However, tools which operate on XML Documents are capable to process the XML representations of the models. So XSLT, GAC and RDL/TT are principally capable of model transformations. For TXL it is also possible, but here the structure must be established first through an according grammar, and also 4DML has first to establish the structure, which actually counts for any type of input for 4DML. The ability for model transformation is poorest available in the UIML peers section. In fact it is not designed for model transformation at all but it allows at least the transformation from the AUI to the CUI and to the FUI. The latter transformation process is the actual goal for the peers section.

Since graph transformations and ATL are designed to work on the model only, they are not capable of processing general XML documents and even less on arbitrary code.[1]

XSLT, RDL/TT and GAC only work with XML Documents and as such are not usable for code transformation as done with TXL, although XSLT and RDL/TT can at least produce non XML code out of an XML Document. The code generation ability is however more evident in 4DML, TXL and UIML.

In the scope of code transformation, UIML peers on the other hand is just able to match UIML elements, but suited very well to produce for different target languages. Besides UIML peers all considered languages allow complex mappings, which means restructuring the source of operation. UIML however only provides a linear one-to-one mapping.

While RDL/TT is both extensible with additional functionality and parameterizable, only few of the other approaches come with these features: ATL allows extensions by calling native operations and GAC is able to process and change context information. While in GAC the context modification happens within the rules, it is separated it in RDL: The context information is fed to variables but processing and modifying the context information is performed with a different rule set.

5 Conclusion

In this paper, we compared several transformation tools / languages with respect to a selected set of criteria we considered of importance for model based user interface

[1] Of course XML document trees can be interpreted as graphs and as such are potentially subject to graph transformations, but we look explicitly at embedded graph transformations as in UsiXML.

development from a more practical view. For this reason we included the programming model, since it may be a premier choice for a developer being familiar with either declarative or imperative programming. We also identified different levels of transformations: Model to model, transformations on XML representation of models and code transformations, while the ability of generating code with a transformation tool is of equal importance to fulfill the complete modeling pipeline. While the capability of specifying complex transformations is very important for the most applications, extensibility and parameterizability is more important to a subset of user interface development tasks, for example for building context-dependent applications.

In respect on the variety of the modeling tasks it is impossible to definitely recommend or dismiss one of the compared candidates which provide different strengths and weaknesses for different applications. For purely model driven approaches, graph transformations and ATL will be good choices but also the XML-processing tools will do, if the model representations come with an XML-syntax.

The most problems can be seen with 4DML since a matrix structure has to be established first, which rather unnatural for user interface models, although it supports indirectly most of the required features. The capabilities of the UIML peers section on the other hand provides only very few features but has proven to be very strong to connect to specific target toolkits.

Therefore, in a short summary, the choice of the transformation tool largely depends on the models, their applied representation and the targeted application. Sometimes, a combination of different transformation approaches is advisable, for example when graph transformations are used for model to model transformations on higher levels of abstractions but the last step towards the final UI requires code generation.

In addition, many user interface modeling tools come with internal transformations which are neither observable, nor controllable. However, their models may be of interest for the user interface developer and are often available with an XML schema. Therefore it makes sense to use the desired models with an own developed ruleset for one of the XML-processing approaches (XSLT, GAC, RDL/TT and to a lesser extent TXL and 4DML) to bypass the tools' internal fixed transformation and for example make transformations to new targets or improve the tools' transformation resuls.

To further compare the performance, code size of the transcoding rule representations, ease of definition and other specific aspects of the transformation tools, more detailed tests are required. Something we did for UIML, XSLT and RDL/TT in the past [16]. Furthermore, the evaluation against the design features developed in [11] is appropriate to get a denser picture and provide user interface developers as well as modeling tool developers a higher level of detail for their choice of transformation approach.

References

1. Abrams, M., Helms, J.: User Interface Markup Language (UIML) Specification, Working Draft 3.1. OASIS (2004)
2. Ali, M.F., Pérez-Quiñones, M., Abrams, M.: Building Multi-Platform User Interfaces with UIML. Multiple User Interfaces - Cross-Platform Applications and Context-Aware Interfaces, pp. 95–118. John Wiley & Sons, Ltd, New York (2004)
3. Andries, M., Engels, G., Habel, A., Hoffmann, B., Kreowski, H.-J., Kuske, S., Plump, D., Schürr, A., Taentzer, G.: Graph transformation for specification and programming. Science of Computer Programming 34(1), 1–54 (1999)
4. Brown, S.S.: Conversion of notations. Technical report, University of Cambridge (2004)
5. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15(3), 289–308 (2003)
6. Cordy, J.R.: The TXL Source Transformation Language. Science of Computer Programming 61, 190–210 (2006)
7. Czarnecki, K., Helsen, S.: Classification of Model Transformation Approaches. OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture (2003)
8. Fiala, Z., Houben, G.-J.: A Generic Transcoding Tool for Making Web Applications Adaptive. In: Proceedings of the CAiSE'05 Forum. CEUR Workshop Proceedings (2005)
9. Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: The Missing Link of MDA. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 90–105. Springer, Heidelberg (2002)
10. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
11. Kay, M.: XSL Transformations (XSLT) Version 2.0, W3C Working Draft. World Wide Web Consortium (2002)
12. Lara, d.J., Vangheluwe, H.: AToM3: A Tool for Multi-formalism and Meta-modelling. In: Kutsche, R.-D., Weber, H. (eds.) ETAPS 2002 and FASE 2002. LNCS, vol. 2306, pp. 174–188. Springer, Heidelberg (2002)
13. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: Usixml: A language supporting multi-path development of user interfaces. In: Bastide, R., Palanque, P., Roth, J. (eds.) Engineering Human Computer Interaction and Interactive Systems. LNCS, vol. 3425, pp. 200–220. Springer, Heidelberg (2005)
14. Nunes, N.J., e Cunha, J.F.: Wisdom - A UML Based Architecture for Interactive Systems. In: Palanque, P., Paternó, F. (eds.) DSV-IS 2000. LNCS, vol. 1946, pp. 191–205. Springer, Heidelberg (2001)
15. Paige, R., Radjenovic, A.: Towards Model Transformations with TXL. In: First International Workshop on Metamodelling for MDA, pp. 162–177 (2003)
16. Plomp, J., Schaefer, R., Mueller, W.: Comparing Transcoding Tools for Use with a Generic User Interface Format. Extreme Markup Languages (2002)
17. Schaefer, R., Mueller, W., Dangberg, A.: Fuzzy Rules for HTML Transcoding. In: Hawaii International Conference on System Sciences, HICSS 35 (2002)
18. Schaefer, R., Mueller, W., Dangberg, A.: RDL/TT, A: Description Language for the Profile-Dependent Transcoding of XML Documents. In: Proceedings of the first International ITEA Workshop on Virtual Home Environments (2002)

19. Soley, R. and the OMG Staff Strategy Group: Model Driven Architecture. White Paper. Object Management Group (2000)
20. Sottet, J.-S., Calvary, G., Favre, J-M.: Towards Model Driven Engineering of plastic User Interfaces. Model Driven Development of Advanced User Interfaces MDDAUI '05. In: CEUR Workshop Proceedings, vol. 159 (2005)
21. Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 16–31. Springer, Heidelberg (2005)