# Towards employing UML Model Mappings for Platform Independent User Interface Design

Tim Schattkowsky
*C-LAB, Germany*
*tim@c-lab.de*

Marc Lohmann
*University of Paderborn, Germany*
*mlohmann@uni-paderborn.de*

## Abstract

*While model based design of platform independent application logic has already shown significant success, the design of platform independent user interfaces still needs further investigation. Nowadays, user interface design is usually platform specific or based on C-level cross-platform libraries. In this paper, we propose a MDA like design approach for user interfaces based on the transformation of UML models at different levels of abstraction. This enables platform independent design of user interfaces and a clear separation of UI and application logic design while enabling full use of native controls in the actual user interface implementation.*

## 1. Introduction

Providing individual User Interface (UI) implementations for each target platform of a contemporary application becomes an increasing burden as the number platforms as well as the size of the applications increases. In the context of model-based design methods and the UML [5], a manual implementation of different UI for each platform is undesirable.

Model Driven Architecture (MDA) [6] presents the idea of mapping a platform independent model (PIM) to a platform specific model (PSM) to separate the core implementation from the platform specific implementation aspects. Still, the UI is often considered to be platform specific although it seems to be possible to provide a generic abstract description of such an UI in terms of a platform independent model.

Furthermore, lack of abstraction in UI design forces large parts of the UI implementation into the responsibility of the software engineer rather than enabling the UI designer to work concurrently. A clear separation of UI and application logic design is desirable to improve both productivity and software quality.

The remainder of this paper is organized as follows: The next section discusses related work before Section 3 introduces our approach. Finally, Section 4 closes with a conclusion and future work.

## 2. Related Work

UI design has been subject to research for quite some time now. However, model based methods are discussed mainly in the context of XML.

The User Interface Markup Language (UIML) [1] is an XML language that aims at providing a meta language for the declarative description of UIs. UIML maps abstract UI elements to actual platform widgets and describes events on these elements. The mapping is based on identifiers with no additional semantics and must be done by the application. UIML does not provide a generic mapping approach from a single abstract specification to different platforms. Furthermore, the event mapping mechanism is quite limited. This is addressed by [2] where a similar UI description is complemented by more sophisticated behavior specification. However, these are note comparable to the expressiveness of UML's behavior models.

The USer interface eXtensible Markup Language (USIXML) [4] addresses the need for more abstraction in UI design, but still in an XML context. However, it introduces the idea to create an abstract UI model based on a domain model that is later refined to a concrete UI model consisting of existing widgets. This model is the basis for generating the final UI implementation. The whole approach is based on XML and graph transformations [3]. It is not aligned with the UML or behavior modeling in general. However, the approach could produce UML compliant output and a UI design tool based on the approach is available [11].

Finally, [7] discusses UI modeling using the UML. Different levels of abstraction exist in the form of a fixed simple model for abstract UIs that is the foundation for manual refinement of the abstract model to the actual application model.

## 3. Design Approach

Our approach is driven by the idea to allow for a complete separation of the UI and application logic design. As in MDA, our approach starts with the creation of a platform independent model. Before generating a platform specific UI implementation, the designer can configure the UI on multiple levels of PIMs, each independent of a target platform. From the most detailed PIM we can generate a platform specific UI implementation (see Figure 1). Transformation rules between the different models facilitate tool support for our UI development approach.

Our basic PIM is the *Information Model (IM)*. This class diagram provides an abstract definition of the information and their logical dependencies. The goal is to develop a UI for presenting this information. It is undistorted by technology information. Therefore, it allows business experts to ascertain much better than with a platform specific model and it provides an early starting point for user interface design.

Figure 2 shows the IM of an administration interface of a simple Web server. The Web `Server` may have a `ContentHandler` at a certain `Port` to which a `Connection` can be made by a `User` to access the content of a `Folder` if he has the necessary `Permissions` according to his `Group` memberships. Furthermore, the pending `Requests` and `Responses` are represented.

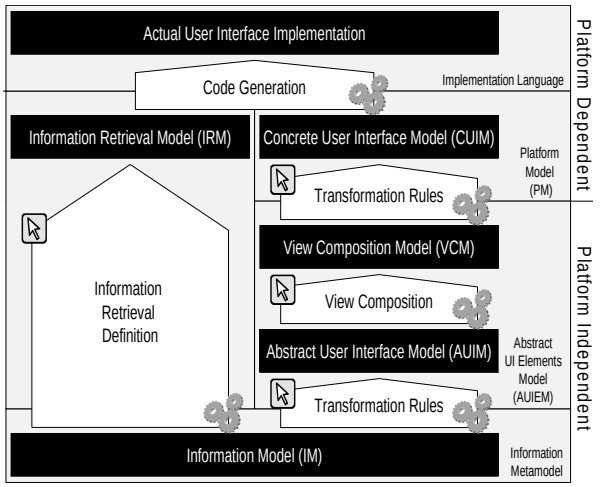


**Figure 1: Platform Independent UI Design Flow**

The associations and attributes in the IM are marked to indicate different kinds of data. Generally, the stereotypes <<readonly>> and <<editable>> mark associations and attributes as only displayable or editable. Attributes marked <<editable>> may have their values altered at runtime while such associations may have links added and removed. If an association is marked as <<deletable>>, links may only be removed in contrast to attributes, which may instead be marked as <<creatable>> indicating that their value may only be by the constructor, i.e., when creating a new instance. If no stereotype is provided for an attribute or association, <<readonly>> is assumed.

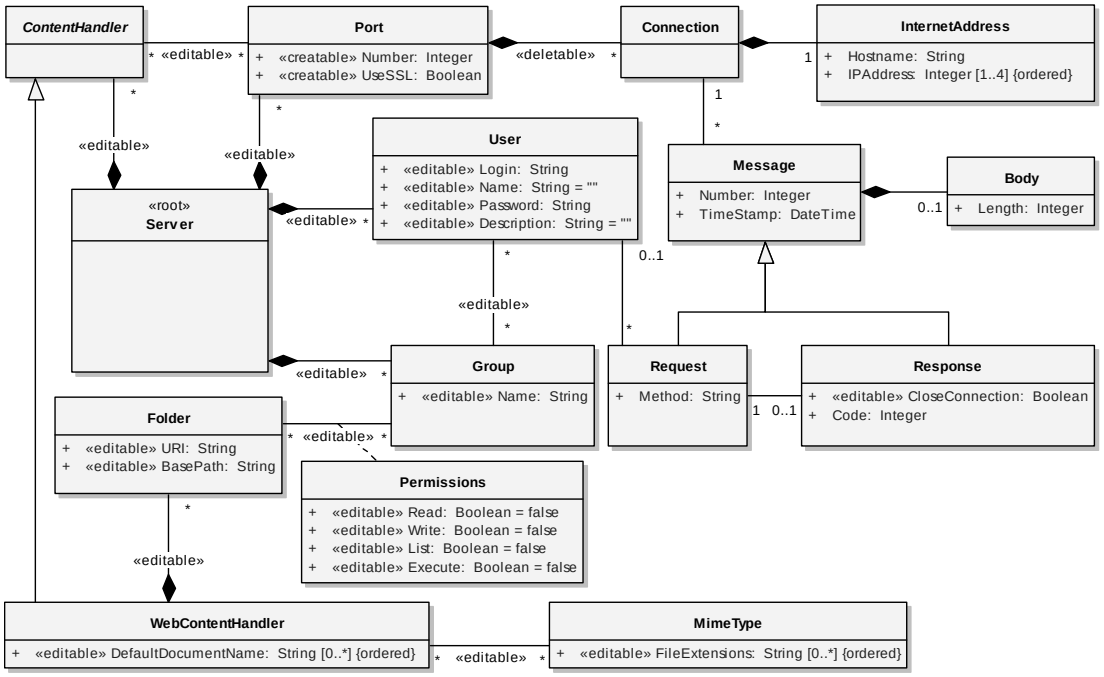


**Figure 2: Web Server Configuration UI Example - Information Model**

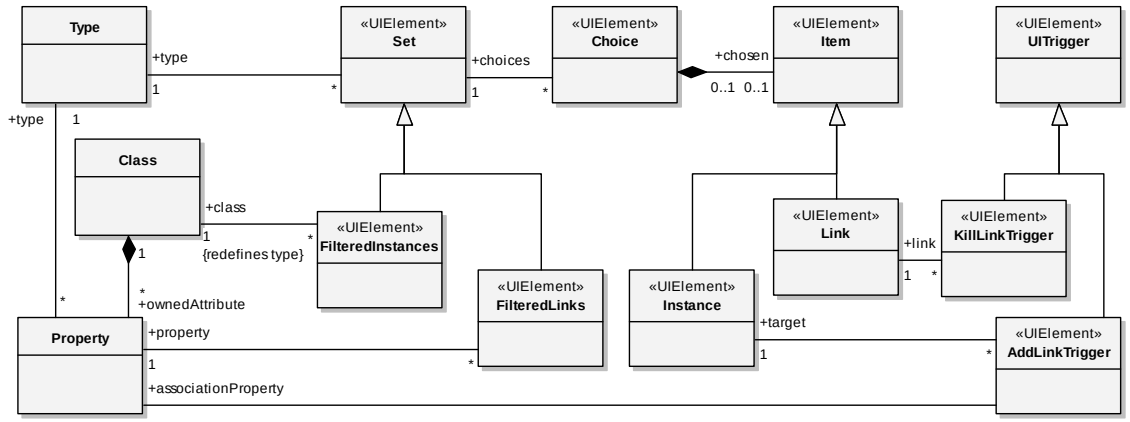


**Figure 3: Excerpt from the AUIEM used for the Example Transformations**

The data types used by the attributes in the IM are fixed and range from primitive types (e.g. Integer, Float) to complex types defined by classes. Operations may define interface application logic that cannot be captured by the data model, e.g., to send explicit messages to the application aside from persistent data.

The whole IM is a composition tree starting by a <<root>> class whose only instance represents the whole systems. This enables inference of aggregations to automatically generate all levels of abstraction from the IM without the need for user interaction. However, usually this is not desirable and the UI designer wants to provide these decisions manually at each level of abstraction.

The PIM at the next level is the *Abstract User Interface Model (AUIM)*. It includes some aspects of UI technology event though platform-specific details are absent. Essentially, the AUIM combines the data from the IM with abstract UI elements to access and manipulate that data. We have developed a metamodel –*Abstract User Interface Elements Model (AUIEM)*– that defines different UI elements at an abstract level in terms of related data sets and triggers. This metamodel can be extended to project-specific needs by using the UML profiling mechanism.

Figure 3 shows an excerpt from the AUIEM employed in our example. This excerpt defines the `Choice` `UIElement` for selecting one `Item` from a `Set`. Furthermore, it provides the necessary elements to employ the `Choice` to select an `Instance` of a `Class` or a `Link` from a `Property`.

These elements are used in a set of graph transformation rules [8] that facilitate tool support for our approach. Each rule consists of a left hand side (subgraph of the IM) and a right hand side (subgraph of the respective AUIM to be created). In Figure 4 an <<editable>> association is mapped to a set of UIElements for deleting and adding links on the association. The basic intuition is that every object or link, which is only present in the right hand side of the rule, is newly created and every object or link, which is present only in the left hand side of the rule, is being deleted. Objects or links which are present on both sides are unaffected by the rule.

The application order of rules is not determined. Furthermore, different rules with the same left-hand side may exist to provide alternative UI elements for the same structure. The actual choice of the desired mappings is an interactive design decision that can be supported by tools. However, complete generation of the AUIM based on the rules is possible. This could be interesting in the context of an UI style defining the actual mappings to be applied.

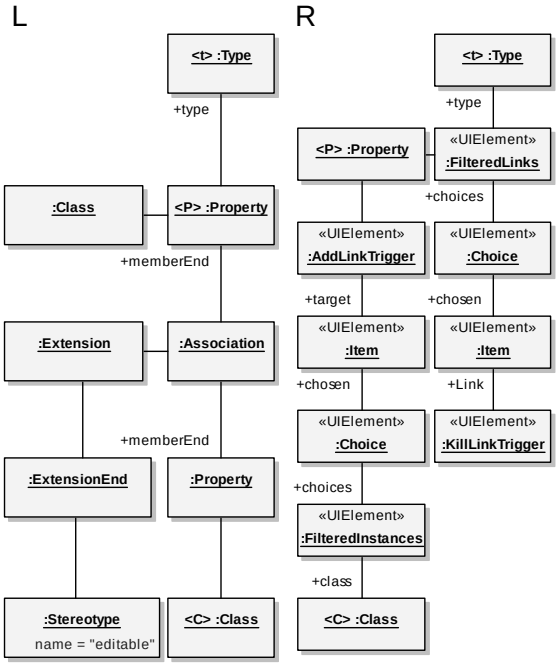


**Figure 4: IM-AUIM Mapping Rule Example**

The most detailed PIM is the *View Composition Model (VCM)*. It partitions the AUIM into several overlapping and navigable views. Each of these views provides the scope of a class instance for the contained UIElements. Thus, master-detail-like views can be implemented. Furthermore, navigation along Links can be defined. Finally, views can be composed. Each contained view either inherits the scope from the containing view or has the scope provided by links selected in the containing view. One root view must be defined. Views enable the purposeful selection of different platform UI elements for the same `UIElement` depending on the overall context of a view while deriving the *Concrete User Interface Model (CUIM)* representing the actual platform dependent user interface.

The CUIM is defined by the *Platform Model (PM)*, which contains a set of available native UI elements on the target platform. Like in the AUIEM, these elements are combined with the elements form the Information Metamodel. Thus, the translation between these models is based on the substitution of the UIElements from the AUIEM by native UI elements from the PM.

The creation of the CUIM not only involves mapping the `UIElements` to actual UI controls (widgets) on the target platform, but also providing additional layout and decoration. A GUI builder tool should support the whole task where the designer may handpick individual mappings for `UIElements`. Transformation rules similar to the rules for the IM-AUIM transformation can be employed here. These rules map `UIElements` and their context to attributed and annotated instances of platform specific UI classes.

The resulting CUIM has to be complemented by the *Information Retrieval Model (IRM)* describing how the data processed by the UI is actually accessed. The IRM is a behavioral UML model (e.g. an activity diagram) giving an operational description how to retrieve the IM elements from the actual implementation. Thus, the IRM functions as an abstraction layer between the UI and the application similar to database abstraction layers. However, the actual implementation of the IRM may vary and is not discussed here.

To create the code for an *Actual User Interface Implementation*, complete code generation takes place combining the IRM and CUIM information to a working platform dependent UI. Again, we use a set of graph transformation rules for the transformation.

## 4. Conclusion and Future Work

In this paper, we have proposed a model-driven design approach for user interfaces based on the UML. This approach allows concurrent development of UI and application logic by starting from a common platform independent information model. Furthermore, due to our code generation mechanisms we can support different target platforms from the same abstract model. The approach has been outlined and discussed in the context of an interface of a Web.server.

We are currently implementing the results of the manual execution of our approach for this example. Future work will include a prototype implementation in the context of our work in the fields of executable models [10] and concurrent software components [9].

## References


[1] Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., Shuster, J. E.: UIML: an appliance-independent xml user interface language. In Computer Networks 31, Elsevier Science, 1999.

[2] Bleul, S., Schäfer, R., Müller, W.: Multimodal Dialog Description for Mobile Devices. In Proc. Workshop on XML-based User Interface Description Languages at AVI 2004, 2004.

[3] Limbourg, Q., Vanderdonckt, J.: Addressing the Mapping Problem in User Interface Design with UsiXML. In Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004, ACM Press, New York, 2004.

[4] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez-Jaquero, V.: UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In Proc. EHCI-DSVIS'2004, 2004.

[5] Object Management Group, The: Unified Modeling Language: Infrastructure. OMG ad/2004-10-02, 2004.

[6] Object Management Group, The: Model Driven Architecture (MDA). OMG ormsc/2001-07-01, 2001.

[7] Pinheiro da Silva, P., Paton, N.: User Interface Modelling with UML. In Proc. of the 10th European-Japanese Conference on Information Modelling and Knowledge Representation, 2000.

[8] Rozenberg, G. et al (eds.): Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1. World Scientific, Singapore, 1997

[9] Schattkowsky, T., Förster, A: A generic Component Framework for High Performance Locally Concurrent Computing based on UML 2.0 Activities. In Proc. 12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), 2005.

[10] Schattkowsky, T. Müller, W.: Model-Based Design of Embedded Systems. In Proc. 7th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC), 2004.

[11] Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005), O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005.