ACTA UNIVERSITATIS AGRICULTURAE ET SILVICULTURAE MENDELIANAE BRUNENSIS
SBORNÍK MENDELOVY ZEMĚDĚLSKÉ A LESNICKÉ UNIVERZITY V BRNĚ

Volume LVII 12 Number 6, 2009

TOWARDS AN MDA-BASED APPROACH FOR DEVELOPMENT OF A STRUCTURAL SCOPE OF THE PRESENTATION LAYER

J. Kryštof

Received: June 26, 2009

Abstract

KRYŠTOF, J.: *Towards an MDA-based approach for development of a structural scope of the presentation layer.* Acta univ. agric. et silvic. Mendel. Brun., 2009, LVII, No. 6, pp. 123–132

This paper presents an approach for developing the presentation layer of software applications. The approach is based on the concept of the Model Driven Architecture (MDA) and uses a UML – based model of graphical user interfaces, which is created according to rules defined in a meta – model. The meta – model is not oriented to a particular platform, thus all designed models can be created independently of the programming language and widget library. This platform independent UML based model can be transformed into source – code for an arbitrary programming language and can be used in a software development process.
The meta – model of our approach is an extension of the common UML and provides support for modeling the presentation layer. The meta – model thus fills a gap that exists in modeling three – layered software applications, beside the application and the data layer. By providing this possibility for modeling the presentation layer, we can crucially impact current approaches to the development of three layered software applications. All model artifacts contain essential information about the graphical user interface and can be used for a code generation. Since the UML is widely used by analysts, they can produce models which de-facto represent source code and thus they reduce the workload for programmers, who create source code by some traditional approaches. Our model – based approach also strictly separates the appearance and the structure of graphical user – interfaces and both of them are developed separately, which brings higher modularity of software.
In this paper, we demonstrate our development approach by focusing on the structure of graphical user interfaces. Our approach is influenced by the concept of Model Driven Architecture and we deal with all related issues, such as meta – model, user models, model transformations and source – code generation. For evaluating our approach, we designed and developed a software framework, we integrated it into a generic modeling tool, and used approach principles during the development of a module of an information system.

MDA, UML, modeling, presentation layer, structure, source code generation

The presentation layer is not a simple homogeneous structure and it's complexity reaches out of frame of this paper. In spite of the complexity of the presentation layer, we can define it as (MYERS, ROSSON; 1992) a software component, that translates a user action into one or more requests for application functionality, and that provides to the user feedback about the consequences of his or her action. The definition refers to all visible components of a software application such as a window, a button or a text section and also refers to an application logic, which accesses controlling units. The application logic can determine all users' actions with their parameters.

When developing the presentation layer, we can follow two traditional approaches (RYDER, SOFFA, BURNETT; 2005): The source code for the visible component can be obtained by hard – coding, or a specialized tool can be used enabling us to get the source code by visual programming. We can identify several drawbacks in both of these, thus we usually use their combination. There still remains one drawback when we use the combination, which is coming from programming itself rather than any technique: the source code is mostly intended for a particular widget library and cannot be used for any other. We characterise this drawback as platform dependency, which can be eliminated by modeling (ZIADI, TRAVERSON, JEZEQUEL; 2002).

The presentation layer is located above the application and the data layer in the software architecture (BACHMANN, BASS, CARRIERE, CLEMENTS, GARLAN, IVERS, NORD, LITTLE; 2005). During the development of the software, the modeling can be intensively employed, so we can use the UML for modeling the application layer or the ERD[1] for modeling the data layer. As we have already mentioned, modeling allows the expression of important information independently of a target platform. Modeling thus can save a lot of work when a current platform needs to be replaced with a different one in order to increase performance or decrease running costs. However, we do not find this kind of support in current modeling tools and thus development is mostly based only on traditional approaches. We perceive this state as a crucial gap and we want to provide a solution for it by designing a model – based approach for the development of the presentation layer.

It is obvious, that creating models is a time consuming activity and our goal is to use them as effectively as possible, and not only as a blueprint or a sketch as with current approaches. If we take into account models from the application or the data layer, we can find in modeling tools support in applications for generating class skeletons with attributes and get/set methods or generation of DDL[2] scripts for a set of target platforms. This automated generation can save time significantly during the development phase, if we consider that in a software system can be thousands of classes or database tables. With respect to these features of automated code generation, we want to have similar support when modeling components of graphical user interfaces (GUI[3]).

MATERIAL AND METHODS

Our research is divided into two parts. The first part focuses on a definition of the modeling concept which includes a modeling language and rules for modeling the presentation layer. In order to model this domain, we can use an existing modeling language or create a new one. The second part focuses on processing model data and transformations resulting in the final model, which will be specific for a particular platform. Furthermore, the second part deals with the issue of source code generation from models.

Choosing the modeling language

In order to provide the modeling concept, we need to choose a modeling language. Firstly we need to define requirements for the modeling language and use an existing language or create a new one. With regard to our goals we define following requirements for a modeling language:

- **Independence the target platform**.
  The modeling language must allow us to express all necessary information without knowledge of any existing platform. We are able to gather dat, which will be valid even after a change of platform.
- **Formalness**.
  In order to transform data from our models, we want the language be formal – thus enabling us to process data automatically.
- **Easy adaptation**.
  We prefer to provide a language which is easy to adopt. We do not want to force analysts or developers to spend time studying a new language, and we prefer to use an existing language with a long tradition, standardized and simple. Users of such a language can directly focus on the real issue and do not have to bear all the difficulties coming from adopting a new language.
- **Expressive power**.
  The modeling language must be powerful enough to describe the target domain. The language should also be flexible and extensible, to reflect possible changes in the target platform.

There have been several modeling languages designed in order to model graphical user interfaces. They provide independence from the target platform and can be processed automatically, so we can choose from these. We can mention UIML (ABRAMS, PHANOURIOU, BATONGBACAL, WILLIAMS, SHUSTER; 1999), which is based on XML[4] and allows us to model user interfaces for windows applications and web browser applications as well. Another similar language is MIMIC (PUERTA; 1996). However, they do not fulfill defined requirements, since they are not widely used and not common. Furthermore, they save model data in a proprietary format so the data can be read and processed only via specialized tools. These facts impact negatively on the usability of the modeling language and the speed of adaptation.

After careful analysis we choose UML, which fulfills all defined requirements and we can declare following:

1 Entity-relationship model – http://en.wikipedia.org/wiki/Entity-relationship_model
2 Data Definition Language – http://en.wikipedia.org/wiki/Data_Definition_Language
3 Graphical User Interface – http://en.wikipedia.org/wiki/Graphical_user_interface
4 Extensible Markup Language – http://cs.wikipedia.org/wiki/Extensible_Markup_Language

- UML provides capabilities to model independently on any target platform so the UML – based models are durable (PILONE, PITMAN; 2005).
- UML is defined formally in its own meta – model with use of the MOF language (Object Management Group, 2005), which implies that UML data can be processed automatically. There also exists a standardization for UML data exchange via XMI[5] format (Object Management Group, 2001): UML models can be exchanged among different modeling tools, which support XMI export/import.
- UML has become an industrial standard as an object oriented modeling language (ENGELS, HECKEL, SAUER; 2000). It has an advantage for developers who have already used this language and do not need to spend a time on learning it. Furthermore, the UML language provides graphical notation which supports novices in adopting it.
- Since version 2.0, UML is enriched by an extension mechanism of UML profiles. This allows us to define restrictions and extensions to this all – purpose language (ABOUZAHRA, BZIVIN, FABRO; JOUAULT, 2005) and tailor the UML to describe a domain of our interest.
- The UML language is also recommend by different methodologies such as the RUP[6] and RAD[7] and its intensive use brings a potential for binding models of all three software layers together, creating a large data base of UML data.

Tailoring the UML for domain oriented modeling

As we have already mentioned in the previous section, the UML provides an extension mechanism in the form of so – called "profiles" for modeling specific domains. UML profiles are used to model those aspects of systems or applications that are not directly describable by native UML elements (ABOUZAHRA, BZIVIN, FABRO, JOUAULT; 2005). A profile is a consistent set of stereotypes, constraints and tagged values. A stereotype represents a class of elements, so we can differentiate between diverse elements of a system. A stereotype can be marked by a geometric icon and can be easily recognized from other stereotypes. While creating our profile, we designed a set of icons and shapes (see fig. 3) in order to help improve orientation when working with large models. Constraints can be attached to a stereotype either in an informal form or in the form of OCL expressions (Object Management Group, 2003). A stereotype is created as an instance of the meta – element of the UML and can have meta – attributes. These meta – attributes are called tagged values and can be used for recording specific properties.

UML profiles have been successfully applied to modeling web systems (KOCH, BAUMEISTER, HENNICKER, MANDEL; 2000), (KARWACZYNSKI, MACIEJEWSKI; 2004) and business processes (JOHNSTON, 2004). With respect to results of these applications, we believe that the mechanism of profiles enables us also to model the scopes of the presentation layer.

UML data processing

In order to generate source code from UML models we need to be able to process UML data. We assume that there is an application programming interface available in the current modeling tool, in a manner as we present in the paper (KRYŠTOF, J., CHALUPOVÁ, N.; 2008). In this case we are able to process data interactively while working with the tool. Another way is to process data which have already been exported to the XMI format. XMI is a language for meta – data interchange and can be applied to all data, which are describable in the language MOF (Object Management Group, 2005). As we have already mentioned, the UML is defined by the MOF, thus we can use the XMI in order to manipulate it.

The UML enables us to create, maintain and develop (SOLEY, 2000) models and the OMG[8] defines a methodical guideline – the MDA (Model Driven Architecture) for processing UML data in software engineering. We deal with issues of the MDA in the paper (KRYŠTOF, 2009a) in detail, so we mention its principles here only briefly.

The MDA assumes the existence of a Platform Independent Model (PIM) created according to some rules in a UML profile. In this case, the UML profile represents a meta – model. By applying the profile on a domain of our interest, we obtain the PIM, which cannot contain any information referring any platform. The PIM stands as an input for model – model transformation, which results in the Platform Specific Model (PSM) that contains information related to a particular platform. The PSM stands as an input for model – text transformation, which results in source code for a chosen platform.

Source code generation

Source code generation is our goal and the last step in our approach to developing the presentation layer. By following the MDA, it is necessary to design a transformation of the PSM into a source code. As we have already mentioned, it is possible to manipulate UML data via the format XMI, thus

5 XML Metadata Interchange
6 Rational Unified Process – http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process
7 Rapid Application Development – http://en.wikipedia.org/wiki/Rapid_application_development
8 Object Management Group – http://www.omg.org/

we can perform the XSLT[9] transformation. However, the complex format of XMI documents and non trivial transformation make writing XSLT programs for code generation difficult and error – prone (SCHAUERHUBER, WIMMER, KAPSAMMER; 2006). We can also use a template – based technique for text generation, as a paper (BOAS, 2004) suggests. Template – based generation is actually nothing new, since it has been used by generations of web pages. We can mention several frameworks which allow the template – based code generation, such as JSP, PHP, Velocity, JET, StringTemplate, etc. Finally, we can create our own specialized program or a library, which will generate text from underlying data. Regarding the wide offer of software libraries for template – based technique and simple usage, we decided to use this approach. The process of transformation is illustrated in figure 1.

Implementation of the approach

We have defined and argued for methods which we use for our model – based development approach of the presentation layer. We can summarize the process of implementation in following steps:

1. Design a meta – model for the presentation layer.
2. Integrate the meta – model in a modeling tool supporting UML.
3. Design transformations PIM – PSM, PSM – source code and create corresponding templates for a particular platform.
4. Evaluate the approach by creating models of a software application, generate source – codes and comment results.

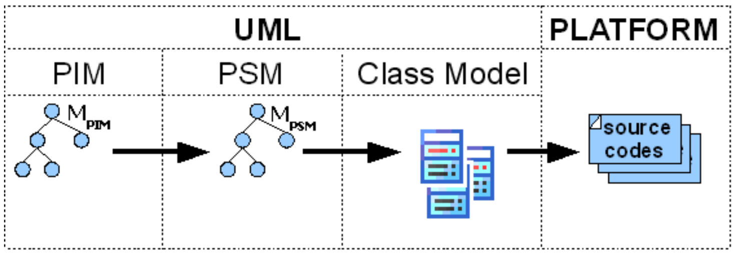
RESULTS

We demonstrate results of the development of a window for a module of an information system (see fig. 6). Since our meta – model is large, we deal only with the structure of user interface and we do not focus on other aspects such as presentation logic, navigation, etc. Detailed information regarding the whole meta – model can be found in our paper (KRYŠTOF, MOTYČKA; 2008).

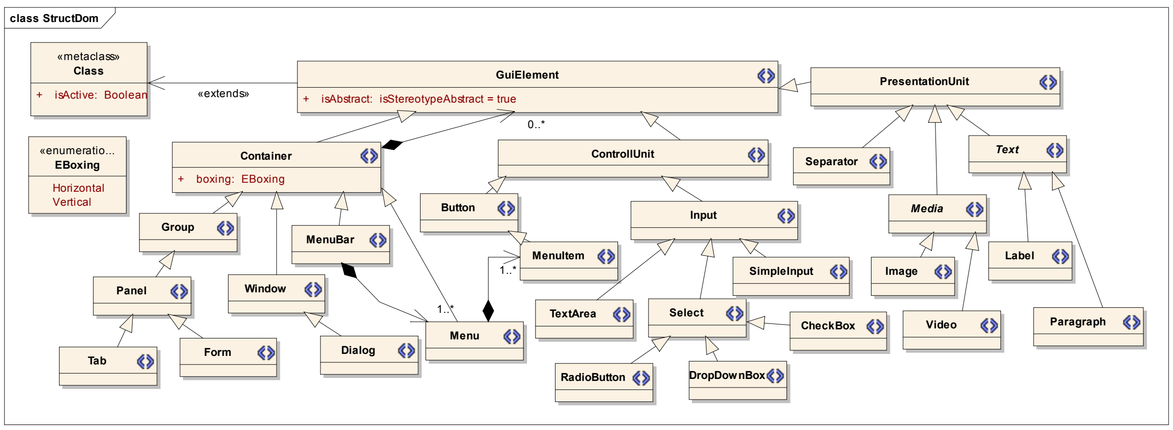
The meta – model of the presentation layer – the structural scope

The meta – model is designed with the use of the UML profile mechanism and has four scope domains: structure, functionality, information and presentation. There exists a corresponding set of stereotypes in our profile for each domain and we defined rules represented by meta – association between them (see fig.4).

The structural domain of the meta – model provides stereotypes corresponding to objects and re-



1: *An overview of model transformation*



2: *The Meta – model of the structural scope*

9 XSL Transformations – http://www.w3.org/TR/xslt

lations, which assemble visible part of the GUI. We use three main abstract stereotypes for modeling visible objects of the GUI: containers, control units and presentation units, which are ancestors of the abstract stereotype "GuiElement" (see fig. 2). Realizations of these abstract stereotypes are for instance "Panel", "Button" or "Text". By finding a proper set of these stereotypes we can model a whole window or its parts. Examples of stereotypes corresponding to relations are "ParentOf" or "Precedes" (see fig. 4). Both relations are connected with the implementation of layout of the GUI. The layout can be implemented in two basic ways (BISHOP, HORSPOOL; 2004): firstly by absolute positioning, when a position of each object is defined explicitly by coordinates, and secondly by relative positioning, when the final position of an object is determined by the logic of its parent container. We implemented relative positioning in our approach, which implies that every object is present in a container. The relation "ParentOf" can exist between a stereotype of the "Container" type, and any other stereotype which is a subtype of the "GuiElement". The relation "Precedes" has two realizations: "H_Precedes" and "V_Precedes", which can exist between any subtypes of the "GuiElement" and defines if objects are placed horizontally or vertically. It is worth mentioning that relations "ParentOf" and "Precedes" are not reflexive and are similar to the "Dependency" relation of the UML. This means that every member of such a relation is either client or supplier. In other words, we can say that the relationship is oriented. We will focus on these relationships in more detail later when we talk about the issue of model transformation.

Meta – model integration into a modeling tool

The meta – model in the form of the UML profile can be integrated into any modeling tool which supports the UML profile mechanism. Regarding our needs, we also require a modeling tool to support import/export to the XMI format or to provide an appropriate API enabling us to access UML data. Nowadays, there exist several modeling tools or frameworks providing these features and we can refer for instance to Sparx Enterprise Architect, Visual Paradigm or EMF.

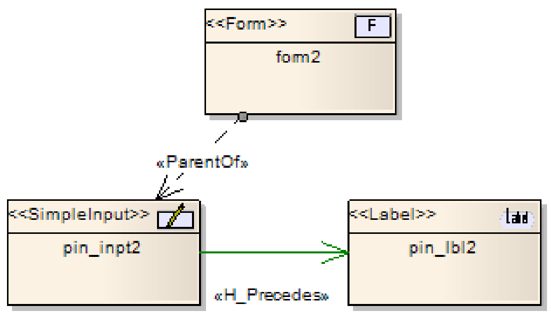
We decided to use the Enterprise Architect (EA) modeling tool and we implemented a set of methods communicating with its API in the .Net platform. We use these methods for validating models and for UML data manipulation, particularly for the process of model transformation. We also integrated our own user interface into the EA, thanks to support from the COM[10] interface. Our user interface serves for launching transformations and code generating interactively, which makes modeling and developing easier and faster. The EA also provides a language called ShapeScript (see fig. 3) enabling us to define shapes and icons for stereotypes. Therefore we designed a set of icons and shapes to create unique appearance of stereotypes in order to make creating and maintaining models more user – friendly.

```
shape main{
  defSize(110, 60);
  rectangle(0, 0, 100, 100);
  addsubshape("xicon", 24, 32);
  addsubshape("xname", 100, 34);
  shape xicon

    editablefield = "stereotype";
    print("<<#stereotype#>>");
    image("form.wmf", 325, 10, 405, 90);

  shape xname

    h_align = "center";
    editablefield = "name";
    moveto(0, 0);
    lineto(100, 0);
    println("#name#");
```

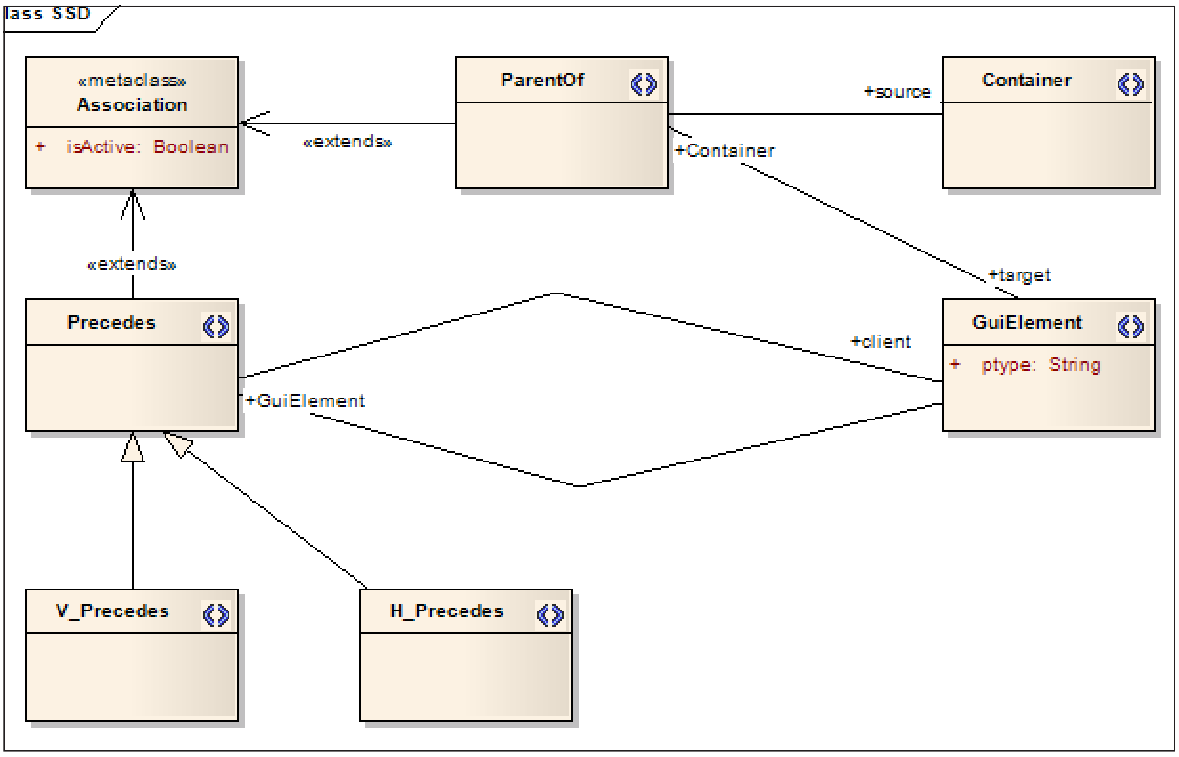


3: *A definition of stereotype appearance in ShapeScript language*

Transformation

In order to generate source code, we need to perform a series of transformations of UML models, so that we get optimal conditions for straightforward and not too complex code generation. Transformations include a model – model transformation in the PIM and the PSM and also model – text trans-

10 Component Object Model – http://www.microsoft.com/com



4: *The meta – model of "ParentOf" and "Precedes" relations*

formation from the PSM to source code. The issue of transformation will be illustrated by transformation of "Precede" relationships.

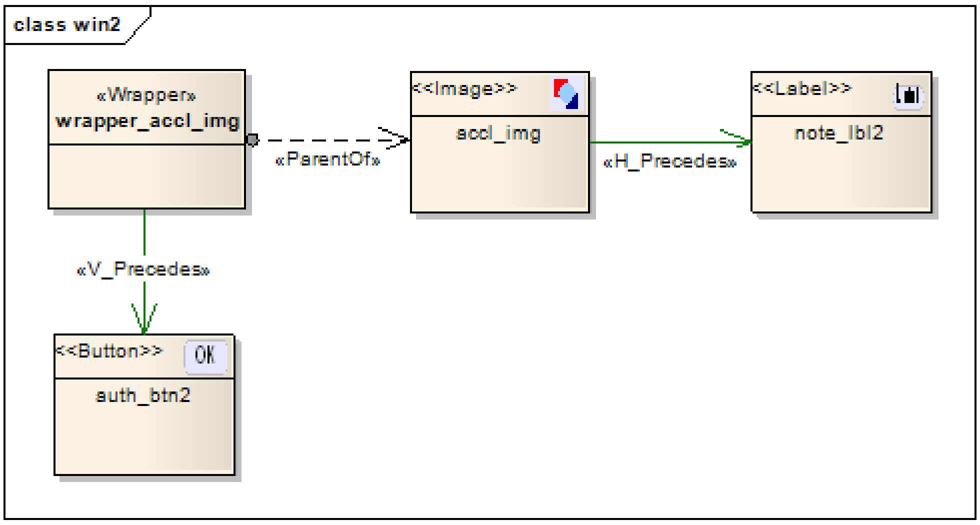
The meta – model allows any element of the GUI to take part in both "H_Precedes" and "V_Precedes" relationships by being in the source role. This state is usually not allowed in a real implementation, since every object must lie in a container. The top – level container is a window or a web page. Containers are usually onedimensional either with a horizontal or a vertical layout: a new member of such a container is laid out next to the last member or below the last member, respectively. Thus no object can be present below/above and next to any other simultaneously in the same container. We have designed an algorithm (KRYŠTOF, 2009b) for removing this unweldom phenomena in models. After the transformation, we get a new model which is enriched by additional "anonymous" containers of type "Wrapper". The model is still platform independent because we did not add any information related to the target platform during the transformation, and we denote the new PIM as the PIM_2. Of course, there can be a need to perform more transformations and the last transformation creates the PIM_N. An illustration of transformation of the "Precedes" relationship is shown in figure 5. The transformation was applied on the stereotype "accl_image" (see figure 6).

The PIM_N is taken as an input for the next transformation, which enriches the current model with information referring to the target platform, so we get the PSM. Our transformation adds tagged values called "ptype", which represents the data type of each stereotype. For this purpose, we created a mapping table, which maps stereotypes to data types in a particular platform. The result of the transformation is a class model, where every class corresponds to a top – level container. Every such class contains a list of attributes, which correspond to all nested objects of the top – level window. Furthermore, class also contains a method "init", which is responsible for initialization of all attributes and placing them into the right containers. This class is subsequently transformed into a source code file and can be compiled by an appropriate compiler.

The last transformation translates the PSM into source code files. The transformation is based on a set of templates, which are designed for a particular target platform. Templates are filled by model data and flushed into a file.

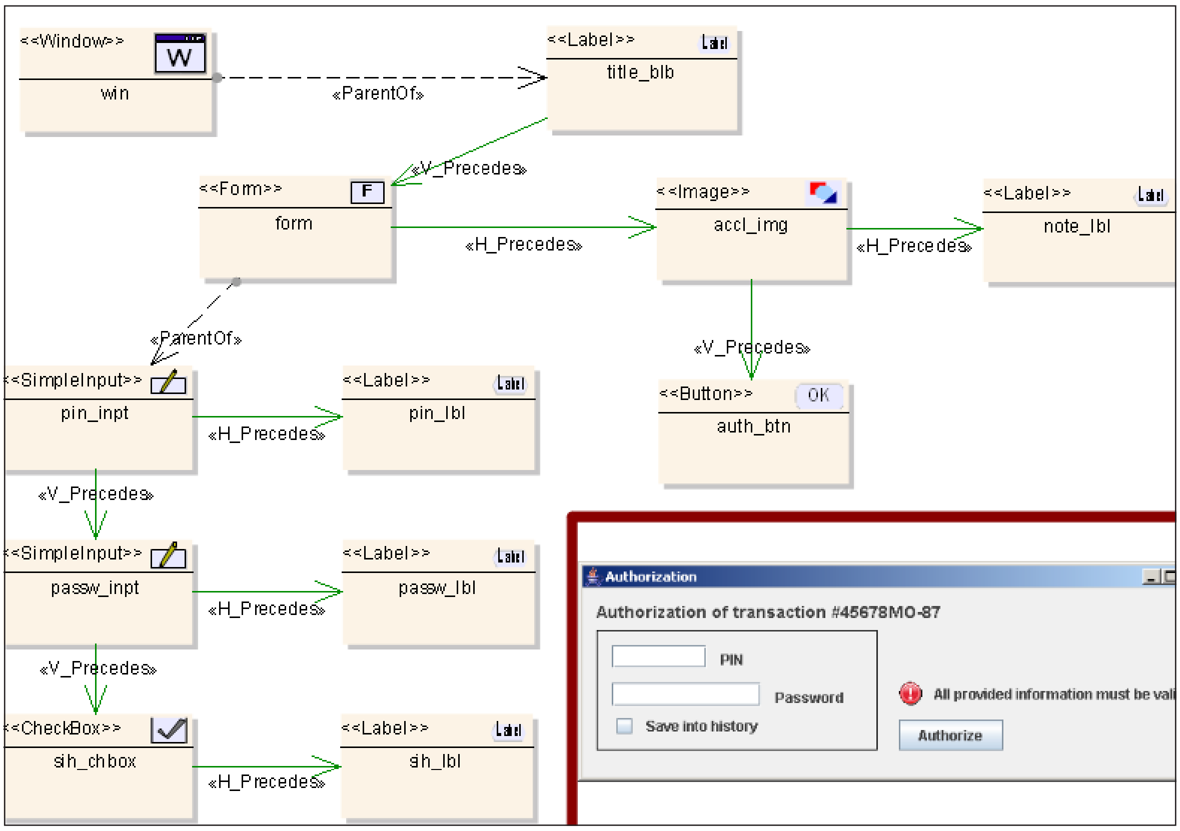
Application of the concept

We have already experimented with our concept by developing a module of a software system. We chose the Java Swing platform as the target widget library and modeled its screens with the use of the profile. After the process of transformation we had a set of files containing source code oriented to building the structure of the GUI. Our modeling approach does not provide any capabilities enabling modeling an appearance of the GUI, thus object properties such as colors or font types are not set. Even the distance between objects is not set properly, so all resulting windows are distorted after execution. We propose separate code responsible for the structure from the appearance in the paper (KRYŠTOF,



5: *An Example of the PIM transfoprmation: normalization of the "Precedes" relationship*

2009). This is actually the same way that web – based applications Mozilla (MCFARLANE, 2003) applications are developed. These applications use separate documents in the CSS format, where the appearance is defined declaratively. In the case of the Swing platform, there is no direct support for CSS technology. Swing is a library written in the Java language, which belongs to a family of imperative programming languages. In imperative languages, the appearance is defined by calling methods for involved objects. It is much easier to write and maintain code for appearance in separate documents rather than in monolithic files. In contrast to Swing's membership in the imperative language family group, we are able to define the appearance in a separate document declaratively! A condition of this approach is a support the reflection mechanism (DUTCHYN, SZAFRON, BROMLING, HOLST; 2001).



6: *An UML profile – based model of a screen*

The reflection mechanism is a kind of meta – programming enabling us to inspect and manipulate objects at run time. We propose to design a document for every generated class that will contain a declarative definition of the appearance, which has a similar syntax to the familiar format of CSS. Basically, the document contains a list of properties with set values. At the moment, when a generated class is executed, we read the file with the appearance definition and inspect particular objects if they provide methods corresponding to listed properties. If there is a match between a property and a method, we can execute the method and provide it an argument – the value of the property in order to set up the appearance. The reflection mechanism is available in some current programming languages such as PHP, Lisp, Net and Java. Since Swing is a library written in the Java programming language, we were able to use the separate appearance definition for generated classes. The idea is illustrated on figure 7.

Using our approach also impacts current work distribution, and moves a significant amount of the workload away from a developer. This work is instead done by analysts and designers in our approach, when they design a specification in the form of a model. In the case of the suggested way of modeling, the model is processed by a software framework, which translates the model to a source code for a particular platform. The resulting source code can speed up work of the developer and presents a benefit which is impossible to get by using blueprints or a wireframes.

Future work

The approach focuses on a model – based development of the structural scope of the presentation layer. We see benefits especially in the possibility to generate source code from the created model of the user interface. We also see the next utilization in providing the model for current software frameworks aimed at the presentation layer, which can use data from this model. With respect to this idea we can think of e.g. the Jakarta Struts framework. Jakarta Struts provides support for generating automated validation data from web forms. The framework needs information about objects corresponding to input fields in order to validate incoming data. Such information is presented in the structural scope of the model and can be provided to the framework in the right format. The right format can also be obtained in the process of transformation. The current approach of providing information related to a particular form relies on manual coding, so the code generation may significantly save time when using the structural model of the GUI.

The meta – model also provides possibilities for modeling the application logic of software applications. With respect to our proposal (KRYŠTOF, CHALUPOVÁ; 2008) it is possible to generate methods which can contribute to the underlying logic of a user interface.

We also believe that intensive modeling of all three layers in the software architecture can provide a rich knowledge base for data mining and deep analysis.

```
XLabel label = new XLabel ("Login");
label.setFontStyle ("Courier", ITALIC, 12);
label.setBackground (Color.green);
label.setMargin (10, 10, 5, 5);
```

```
*
  font: Tahoma;
  foreground: black;

XLabel
  margin: 10px, 10px, 5px, 5px;
  font-style: courier, italic, 12;
  background-color: green;
```

7: *Replacing an imperative appearance definition by a declarative definition*

SUMMARY

We introduced an approach for model – driven development of the structural scope of the presentation layer. The approach strongly relies on modeling and its implementation is influenced by the Model Driven Architecture concept. We created a UML profile in order to model four scopes of the presentation layer. The profile provides rules for creating and validating models and can be extended in the case of any need.

A model of the presentation layer is used for a series of transformations which modify the initial abstract model and finally create a very concrete model with platform specific information – the PSM. The form of the PSM enables a straightforward and simple generation of source code for a particular platform. The PSM is created as a class model and every class corresponds to a top – level container, which is usually a window or a container, e.g. a form or a menu. Every class has a list of attributes corresponding to all nested objects of the container and has a method "init" which initializes all nested objects and places them into the correct parent containers. The PSM is taken as an input for the last transformation, which produces a set of files containing a particular source code. The source code does not contain any information related to the appearance, so the appearance is defined in a sepa-

rate file in a declarative language. In order to set the appearance in non – declarative language we use a reflective way of programming, so we are able to set the appearance even for an imperative language such as Java.
The approach brings several benefits including an efficient creating specification of the presentation layer in the phase of analysis, a platform independent development and highly maintainable code of the appearance definition. The specification in the form of a model is directly used in the development phase, when a developer starts his or her work by generation of the structural scope of the GUI. This speeds up the starting phase of the project and save time for other tasks. Furthermore, keeping information related to the appearance and the structure separately makes source codes easy to maintain and the appearance can be changed without recompiling any source code file.
The approach fills a logical gap in modeling three layered software applications with the use of UML, since current modeling tools provide support for modeling the presentation and data layers only. We implemented a support for modeling the structural scope of the presentation layer with the Sparx Enterprise Architect modeling tool and we also provide features for code generation.

SOUHRN

Využití konceptu MDA ve vývoji strukturální oblasti prezentační vrstvy

V článku představujeme modelem řízený přístup vývoje prezentační vrstvy. Metodický postup je ovlivněn konceptem modelem řízené architektury a my jej uvádíme do souvislosti s problematikou modelování grafických uživatelských rozhraní. Pro modelování využíváme profil UML, který jsme pro naše potřeby modelování vytvořili. UML profil poskytuje pravidla a prostředky pro vytváření modelů prezentační vrstvy, takže je možné nejenom modely vytvářet, ale je možné je i validovat.
Ručně vytvořený model prezentační vrstvy používáme jako vstup pro sérii transformací, které model obohacují. Transformace na úrovni modelu je ukončena stavem, kdy je vůči svrchním komponentám vytvořen model tříd, který využíváme pro generování zdrojového kódu.
Zdrojový kód svrchních komponent obsahuje deklarace všech prvků, které se na tvorbě svrchní komponentě podílejí. Dále obsahuje metodu, která zajistí jejich umístění do kontejnerů. Vygenerovaný zdrojový kód není zodpovědný za nastavování vzhledu. Vzhled nastavujeme ke každé svrchní komponentě zvlášť a v případě, že cílová platforma nepodporuje mechanismus CSS, využíváme k nastavování vlastností vzhledu reflexe.
Náš koncept přináší několik výhod, mezi které patří platformní nezávislost modelů a tedy i možnost generovat zdrojové kódy stejných komponent pro jiné platformy. Možnost generování kódu může taktéž významným způsobem zrychlit práci a ušetřit tak pracovní kapacity.
Dále náš koncept zaplňuje logickou mezeru v modelování třívrstvých aplikací, jelikož nabízíme prostředky pro modelování a zároveň jsme schopni generovat zdrojový kód tak, jak je tomu u UML modelovacích nástrojů, které takto podporují aplikační a datovou vrstvu. Intenzivní modelování všech tří vrstev a jejich soustředění v modelovacím nástroji může vytvořit zajímavé podmínky pro získávání znalostí o aplikaci.
Naše modelovací prostředky záměrně neumožňují generovat kód nastavující vzhled. Vzhled aplikace je vyvíjen a spravován samostatně, což má následek snadnější správu vzhledu jako celku a lze tedy vzhled aplikace měnit bez zásahu do zdrojového kódu. Možnost takovéto kustomizace pak může výrazně zvýšit zájem o softwarový produkt a zvýšit konkurenceschopnost softwarové firmy.

MDA, UML, modelování, prezentační vrstva, struktura, generování zdrojového kódu

The paper is written as a part of solution of a research plan PEF MZLU MSM 6215648904/03/03/02.

LITERATURE

MCFARLANE, N.: Rapid Application Development with Mozilla. Prentice Hall 2003, 800 pages, ISBN 978-0131423435.

ABOUZAHRA, A., BZIVIN, J., FABRO M. D. D., JOUAULT, F., 2005: A Practical Approach to Bridging Domain Specific Languages with UML profiles. In: Proceedings of the Best Practices for Model Driven Software Development at OOPSLA'05, 2005.

ABRAMS, M., PHANOURIOU, C., BATONGBACAL, A. L., WILLIAMS, S. M., SHUSTER, J. E., 1999: UIML: an applianceindependent xml user interface language. In: Computer Networks 31, Elsevier Science, 1999.

BACHMANN, F., BASS, L., CARRIERE, J., CLEMENTS, P., GARLAN, D., IVERS, J., NORD, R., LITTLE, R., 2005: Software Architecture Documentation in Practice: Documenting Architectural Layers. Technical Report CMU/SEI-2000-SR-004, Carnegie Mellon Software Engineering Institute, 2005.

BISHOP, J., HORSPOOL, N., 2004: Developing principles of gui programming using views. In: SIGCSE

’04 Proceedings of the 35th SIGCSE technical symposium on Computer science education, 2004.

BOAS, G., 2004: Template Programming for Model-Driven Code Generation, [online], [cit. 2009-06-17]. Available at WWW: <http://www.softmetaware.com/oopsla2004/emdeboas.pdf>

DUTCHYN, C., SZAFRON, P. LU, D., BROMLING, S., HOLST, W., 2001: Multi-dispatch in the Java Virtual Machine design and implementation. In: COOTS’01 proceedings, 2001.

ENGELS, G., HECKEL, R., SAUER, S., 2000: UML – a universal modeling language? ICATPN 2000, LNCS 1825, Springer-Verlag. 2000.

JOHNSTON, S., 2004: Rational UML Profil for Business Modeling. [online], 2004 [cit. 2009-06-17]. Available at WWW: <http://www-128.ibm.com/developerworks/rational/ library/5167.html#author1>

KOCH, N., BAUMEISTER, H., HENNICKER, R., MANDEL, L., 2000: Extending UML to Model Navigation and Presentation in Web Applications. In: Proceedings of Modelling Web Applications in the UML Workshop, 2000.

KRYŠTOF, J., CHALUPOVÁ, N., 2008: Prerequisites for new GUI modelling approach. In: JANECH, J. Objekty 2008. Žilina: Žilinská univerzita v Žilinie, 2008, p. 127–136. ISBN 978-80-8070-927-3.

KRYŠTOF, J., MOTYČKA, A., 2008: Metamodel for presentation layer, In: Proceedings of the 11th International Multiconference Information Society – IS 2008. 2008.

KRYŠTOF, J., 2009: An automated platform independent realization of GUI with use of UML. In Gaudeamus. IMEA 2009. Hradec Králové: Univerzity Hradec Králové, 2009, p. 14–18. ISBN 978-80-7041-851-2.

KRYŠTOF, J., 2009: Formal describtion of layout in graphical user interfaces. In: 11th International Conference MEKON 2009. 1. vyd. Ostrava: VŠB – TUO, Faculty of Economics, 2009, ISBN 978-80-248-2013-2.

KRYŠTOF, J., 2009: Impact of the Model Driven Architecture on Competitiveness of Software Producers. In: ŽUFAN, P. Firm and competitive environment. 2009 – 5. část. Brno: MSD, s. r. o., 2009, p. 61–65. ISBN 978-80-7392-088-3.

MCFARLANE, N., 2003: Rapid Application Development with Mozilla. Prentice Hall 2003, 800 pages, ISBN 978-0131423435.

MYERS, B. A., ROSSON, M. B., 1992: Survey on user interface programming. In: Proceedings of SIGCHI’92, Monterey, California, 1992.

Object Management Group: MOF 2.0/XMI Mapping Specification v2.1 [online], 2009 [cit. 2009-06-17]. Available at WWW: <http://www.omg.org/docs/formal/05-09-01.pdf>

Object Management Group: UML 2.0 OCL Specification. [online], 2003 [cit. 2009-06-17]. Available at WWW: < http://www.omg.org/docs/ptc/03-10-14.pdf.

Object Management Group: UML 2.0 Superstructure. [online], 2005 [cit. 2009-06-17]. Available at WWW: <http://www.omg.org/spec/UML/2.0/Superstructure/PDF>.

PILONE, D., PITMAN, N., 2005: UML 2.0 in a Nutshell. CA, USA: O Reilly, 2005. ISBN 0-596-00795-7.

PUERTA, A., 1996: The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996.

RYDER, B. G., SOFFA, M., BURNETT, M., 2005: The impact of software engineering research on modern progamming languages, ACM Transactions on Software Engineering and Methodology (TOSEM), 2005.

SCHAUERHUBER, A., WIMMER, M., KAPSAMMER, E., 2006: Bridging existing Web modeling languages to model-driven engineering: a metamodel for WebML, Workshop proceedings of the sixth international conference on Web engineering, 2006.

SOLEY, R., 2000: Model Driven Architecture. [online], 2000 [cit. 2009-06-17]. Available at WWW: < http://www.catalysis.org/publications/papers/2001-mda-Overview-00-11-05.pdf>

ZIADI T., TRAVERSON B., JEZEQUEL J., 2002: From a UML Platform Independent Component Model to Platform Specific Component Models, In: Proceedings of Workshop in Software Model Engineering, Fifth International Conference on the Unified Modeling Language, 2002.

Address

Ing. Jan Kryštof, Ústav informatiky, Mendelova zemědělská a lesnická univerzita v Brně, Zemědělská 1, 613 00 Brno, Česká republika