An LPGM Method: Platform Independent Modeling and Development of Graphical User Interface

Jan Kryštof
Faculty of Business and Economics, Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic
E-mail: jan.krystof@mendelu.cz

Keywords: GUI, HCI modeling, MB-UID, UML, adaptive modeling tool

Received: September 12, 2009

This paper introduces a new method in the area of platform independent modeling and the development of graphical user interfaces. The method bridges the gap between traditional MB-UIDEs and the modern web methodologies by enabling the modeling and development of both traditional and web user interfaces. The method is based on a proposed Presentation model and a Task Action Model which drive the development process. The modeling notation in both models is done with use of UML, and the development process is supported by a UML-compliant adaptive modeling tool. Descriptions of both the model and the method of application are included. An evaluation done using a JavaEE and a Swing widget toolkit is also mentioned.

Povzetek: Predstavljena je nova metoda za izdelavo platform za razvoj grafičnih vmesnikov.

1 Introduction

In the course of developing software for a user interface (UI), a developer frequently recognises that a similar UI has been created previously, perhaps in a different context and with different visual aspects, but nearly identical in concept. This research investigates the possibilities of reusing UIs.

UIs can be made more readily reusable by elaborating the specifications for them in a form that is independent of platform. Such platform-independent tools and methodologies have been developed, but, unfortunately, the results have never achieved widespread adoption and successful application in industry [12, 38, 17].

UI development is a difficult and time-consuming procedure [37] that involves a collection of different activities. UI development deals with the interaction between humans and computer and specifies how software will function across this i.e. the tasks of the user and the system. The physical user interface is subsequently assembled with respect to the tasks identified for the user and the system. The UI should have appropriate ergonomics and appearance and it must communicate with the underlying application layer. The process of UI development is not properly described in traditional software development methodologies (Waterfall, Spiral). For example, the Unified Process provides advice for UI development only by recommending the build-up of a prototype [19]. The UI prototype in the Unified Process serves only as a tool for better understanding the particular use-case and its functionality. The methodological framework Rational Unified Process [28] goes a step further, extends the number of artifacts used for UI modeling, and introduces the UI storyboards [43]. However, neither the UP nor the RUP adds methodological guidelines for UI development.

The lack of development guidelines for the UI development was partially covered in traditional methodologies by the concept of Model-Based User Interface Development (MB-UID), which unified development of applications using a traditional UI. The development is based on intensive modeling of the different aspects of all part of the application, including aspects of UIs. With the emergence of web-based development, a number of new web methodologies have been proposed [23, 7, 55, 50]. These define the entire development process for web applications, including issues of UI development. However, these modern web methodologies and MB-UID represent disparate branches, which can be used for either traditional or web user-interface development.

In this paper a new approach for UI modelling will be presented along with the architecture of development environment for this approach. The approach aims to provide a method for producing platform-independent modelling and development of graphical user interfaces using pure UML. The following section will deal with the current state of MB-UID and some of the drawbacks which led to this new approach. The section three describes the approach and later in section four, steps for applying the approach will be presented. Some examples will be included in the section five.

2 Current state

2.1 Model based user interface development

Systematizing of UI development is a challenging and important prerequisite for the quality of development, and the concept of MB-UID supports it. MB-UID is characterized by a set of declarative models and a way of interpretating them [48]. The MB-UID approach is supported by software environments which are called MB-UIDEs (Model-Based User Interface Development Environments).

The UI development process is focused purely on constructing models which describe different areas of application. Models are built incrementally, describing "what" without explaining "how", thus hiding the method of implementation. However, approaches within MB-UIDs are not yet mature, and proposals for the range and nature of the models supported differs significantly [17]. The development process varies with environment, since each particular MB-UIDE defines its own set of models. Thanks to this diversity, the models mostly commonly encountered are [53, 48]: the domain model, the application model, the task model, the user model and the presentation model.

Many different notations are used for MB-UID, because no uniform standard for all MB-UIDEs exists. In general, notation has been developed specially for each the MB-UIDE [53] which makes it difficult for developers to get oriented in other forms of notations and causes compatibility problems: a model created using a particular tool can not be processed using a different tool. Silva divided MB-UIDEs into two generations [53]. Second generation environments are oriented more towards industrial standards and are more receptive to new user-interface features. Despite enhancements, interoperability remains rather low, and the MB-UIDEs are not in widespread use among developers. There are also addressed two more drawbacks of the MB-UID approach [38]. Firstly, the generated UIs are often not as good as those that could be created using conventional programming techniques. Secondly, heuristics are often involved and the connection between the specification and the final result can be quite difficult to understand and control. This makes the results unpredictable. We assume that efforts to generate "final" and "ready to run" products cannot succeed and make extension of any particular MB-UIDE to support a new platform very hard. The reusability of models is associated with the whole application of MB-UIDEs, so it is not possible to make use of a single model. With regard to the specification of a presentation model in MB-UIDEs, we can address one significant drawback which is connected with the separation of concerns [39, 15, 26]. Concerns are often merged together with visual appearance, layout or content specified within a single presentation model, which makes such a form useful only for the original requirements. Furthermore, the layout of UIOs is sometimes specified in terms of the absolute positioning [34]; this is the possibility, that constraints of screen and resolution will prevent the proper display of the user interface.

2.2 Modern web methodologies

A similar approach to the generation of applications in development is driven by modern web methodologies such as OOHDM [50], WebML [7], UWE [23, 24] or OOWS [14]. These also provide methodological guidelines for specifying sets of declarative models which drive subsequent development. Therefore they fit the concept of MB-UID. In order to make a clear distinction between web and traditional development, we will use the terms "web MB-UID", and "traditional MB-UID" respectively. Some of these methodologies (UWE, WebML, OOHDM) also provide software environments (ArgoUWE [22], WebRatio [58] and OOHDM-Web [49]) in order to support the modeling approach by means of a set of frequently used functions in the context of model construction or code generation. Thus we can classify them as web MB-UIDEs.

Like traditional MB-UIDEs, web MB-UIDEs suffer from low interoperability since they also use their own modeling notation, which makes the interchange of model data between different environments impossible. On the other hand, some web methodologies have already employed UML for modeling notation. UML is the de-facto industrial standard object-oriented modeling language [13]. The notation is familiar to many developers, and there are a lot of resources such as documentation and software support in the form of modeling and CASE tools. The UML profiles mechanism [40] is also used sometimes to provide new modeling facilities. Since UML profiles are based on UML, it is not difficult for any software designer with a background in UML to understand a model based on a UML profile [24]. Regarding the summary of the modeling notations employed in web methodologies published in [11], the UML notation is fully employed in OOWS while some other methodologies (e.g. OOHDM, UWE, WebMl) combine UML with other forms of notation (e.g. OO, OMT, ERDs, DFD), and the rest do not use UML notation at all. The set of declarative models is nearly the same in web methodologies compared to the model sets in MB-UIDs, except for the navigation model which is tightly connected with the hypertext paradigm. The MDA (Model Driven Architecture) [16] concept is used in some environments (ArgoUWE, WebRatio) in order to interpret models and support code generation.

2.3 Characteristics of traditional and MB-UID in summary

From the overview that has been carried out the preceding sections, we want to point out several positive and negative characteristics of current MB-UID.

Model-based development. The UI development in MB-UIDEs and web methodologies is based on the construction of different models. Employing modeling approaches in UI development has proved that modeling represents a good way to support user-interface development.

Development environment. MB-UID is supported by graphical software environments which enable rapid model construction and utilization of models. Using graphical environments also reduces the cognitive burden on human programmers [47].

Diversity of modeling notations. Many varied notations employed in different approaches do not support interoperability, since data obtained from the model can be read and modified only in the original environment.

Heavy-weight solutions. Solutions produced in MB-UID tend to provide "ready to run" software. Attempts to cover the presentation, application and data layer which result in the great complexity of traditional MB-UIDEs and make any extension of such environments difficult.

Low separation of concerns. The separation of concerns in presentation models is very low both at the model and source code levels, which impacts the reusability. Thus models can hardly be used on different platforms.

Disjunction of development of web and traditional user interfaces. Modern web methodologies and traditional MB-UID have built two disjoint branches that focus on either web or traditional UI. There is no middle ground where both scopes can be developed.

3 Modeling approach

The summary mentioned in the previous section has contributed to forming our approach to the scope of MB-UID: our approach is built with the respect to the pros and cons of the current state of MB-UID. In this section, our modeling approach is introduced and described along with an argument for chosen methods which are involved.

The goal of our approach is to provide facilities enabling the modeling and development of platform-independent user interfaces. In order to provide appropriate facilities for modeling user interfaces and user-computer interaction, we need to choose a modeling notation which enables us to model these domains. Basically, we can design our own notation and create a domain-specific language (DSL) [1, 54]. This approach is recommended for cases where the modeling domain is large, the modeling area is well charted, and there is only a small probability of further evolution. The main disadvantage of the DSL approach is that in setting up a new notation we might be considered to adding another stone to the tower of Babel of modeling languages. Furthermore, we want to have a language that is easily extensible, since the field of user interfaces is constantly evolving. Therefore we chose an approach employing UML profiles – a light-weight extension of UML [47] which preserves compatibility with UML. The UML profiles mechanism is currently the most utilized approach thanks to the large number of CASE tools [1] and other support (UML-compliant tools, extensive documentation, and a wide base of users) available for it. Many projects [20, 23, 9] have employed UML profiles to model diverse domains and benefited from the high level of interoperability, thanks to the XMI (XML Metadata Interchange) data format [41]. For these reasons, we chose the UML and its profiles to define and provide modeling facilities, and we created a UML profile called "LPGM" (Lightened Profile for GUI Modeling), which gave its name to our approach. This profile is presented in fig. 3 and fig. 4.

Our profile provides facilities for building two kinds of models: the task-action model focused on aspects of the interaction and the presentation model focused on the structure of the user interface.

3.1 Task-action model

Since all user interfaces are associated with an underlying application layer that performs a particular business [36], we wanted to link the user-interface to it and express the business in terms of the user-interface interaction.

The functionality conveying associated business is usually specified in the form of a use-case model and its documents. The user-interface interaction is also recorded here in the text form of scenarios. Scenarios often include references to particular interaction objects of the relevant user interface [43], as is common in use-case in the Rational Unified Process [28].

Many styles of writing scenarios (common narrative style, partitioned narratives, pseudo-code, interaction diagrams, etc.) are summarized in [10]. The most comprehensive style was formed by Wirfs [59]; in it the scenario is captured in a two-column dialog between the system and the user. This style of interaction capture is very natural, since the user-interface interaction is a kind of dialog consisting of the user's action and the system's reaction.

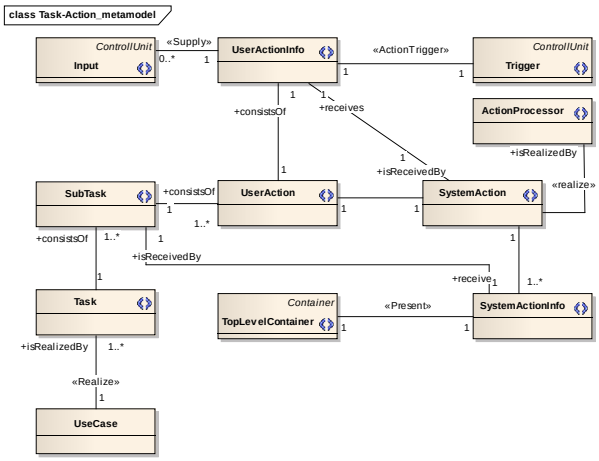

Figure 1: Task-Action Meta-model.

In order to specify the user-interface interaction, we synthesized a two-column dialog scenario capture with a

behavioral diagram and formed the Task-Action Model (TAM) [31]. The goal of TAM is to convey the user-interface interaction by capturing 1) the goal of the user, 2) the user's responsibility to the interface and 3) the system's responsibility to the interface.

We chose the UML Activity diagram as modeling facility because of its simple notation compared with common interaction diagrams. A UML activity diagram is normally used to represent the dynamic view of a system as control and data flow from activity to activity [6]. In our case we have used it to depict the flow of actions performed by the user and the system. The Activity diagram has also been successfully employed in user-interface storyboarding in RUP [43] and it has been proposed as a suitable diagram for CTT (Concur Task Trees) [3, 42], a widely used notation for task modeling. However, we want to model tasks in the context of the user and the system to show how these tasks should be performed in terms of elementary actions as well as to show which data are transferred during the steps of interaction. The TAM, specified in the meta-model shown in fig. 1, is based on our proposed meta-model for the presentation layer [32]. All of the meta-model elements are described in the table 1. The TAM is commonly constructed after analysis of a particular use-case where at least one task having a goal has been identified. We consider the terms “task” and “goal” as they are defined in Hierarchical Task Analysis (HTA): a task is an activity that a user does to reach a goal, while the goal is a desired state of the system [21]. Each task can be broken down into several subtasks. Each subtask has associated with it a particular container which represents a set of user interface objects (UIO). The subtask is a composition of one or more atomic actions which are associated with particular interaction objects (IO). An action associated with a subtask is called a User action and denotes a user responsibility with the respect to one or more IOs. We model two kinds of interaction: 1) Supply interaction, which represents providing input data for a current task and 2) Trigger interaction, which causes termination of a current subtask and transition to a connected System action. The System action is

Meta-model object	Description	UML	Location
UseCase	Use-case associated with one or more task.	Original	Use-case model
Task	The task is bound to a particular use-case through a dependency «Realize». The task has one or more SubTasks.	Activity, stereotype «Task».	Task Action Model
SubTask	The Subtask represents one or more steps which belongs together within a task. It has an input (SystemActionInfo) which holds a reference of UI displayed within this subtask. It has one or more UserActions.	Activity, stereotype «SubTask».	Task Action Model
UserAction	The UserAction represents a user-interface interaction which has one UserActionInfo.	Action, stereotype «UserAction».	Task Action Model
UserActionInfo	The UserActionInfo is the specification of a particular UserAction and conveys more information about the interaction. The UserActionInfo can have one or more UIOs of the ControlUnit (from LPGM structural model) type associated through «ActionTrigger» or «Supply» dependency. The UserActionInfo. This object is received by a SystemAction which processes the UserAction.	ActionPin, stereotype «UserActionInfo».	Task Action Model
Input	An input object (TextField, CheckBox, etc. from LPGM structural profile) used during a user-interface interaction for obtaining data from a user. It is connected with the UserActionInfo through the «Supply» dependency.	Class, stereotype «Input» and its descendants.	Presentation model
Trigger	The object (from the LPGM structural model) used during a user-interface interaction for triggering a SystemAction. It is connected with a UserActionInfo through the «ActionTrigger» dependency.	Class, stereotype «Trigger» and its descendants.	Task Action Model
Supply	The dependency between a user and UserActionInfo and a particular Input object (e.g. TextField, CheckBox). It denotes the user's responsibility for providing data to the current SubTask.	Dependency, stereotype «Supply».	Task Action Model
ActionTrigger	The Dependency between a UserActionInfo and a particular Trigger object (e.g., Button, MenuItem). It denotes a user operation which terminates the current SubTask.	Dependency, stereotype «ActionTrigger».	Task Action Model
SystemAction	The SystemAction represents an abstraction of the system action responsible for processing the previous SubTasks. It is responsible for processing the previous SubTask and providing a UI as a response.	Action, stereotype «SystemAction».	Task Action Model
SystemActionInfo	The SystemActionInfo is a specification of a particular SystemAction. It holds a reference to a method ActionProcessor and TopLevelContainer that is generated and displayed in the subsequent task.	ActionPin, stereotype «SystemActionInfo».	Task Action Model
ActionProcessor	The ActionProcessor is a method which represents a physical implementation of the SystemAction. It is responsible for processing the data provided by the previous SubTask.	Operation, stereotype «ActionProcessor».	Application model
TopLevelContainer	TopLevelContainer is a UIO which is generated as a response and passed to the subsequent SubTask.	Class, stereotype «TopLevelContainer».	Presentation model
Presents	The dependency between SystemActionInfo and generated UIO. The dependency between the user and UserActionInfo and a particular Input object (e.g., TextField, CheckBox). It denotes the user's responsibility for providing data to the current SubTask.	Dependency, stereotype «Presents».	Task Action Model

Table 1: Description of the Task-Action Meta-model.

responsible for processing the finished subtask through a delegated method denoted as the Action processor. This method generates a user interface which is represented by a container. One interaction step is finished at this point and a new one begins by sending the generated container to the following subtask. The whole process is repeated until the last subtask is finished and the goal associated with the current task has been achieved. Example of the TAM is presented in the fig. 2.

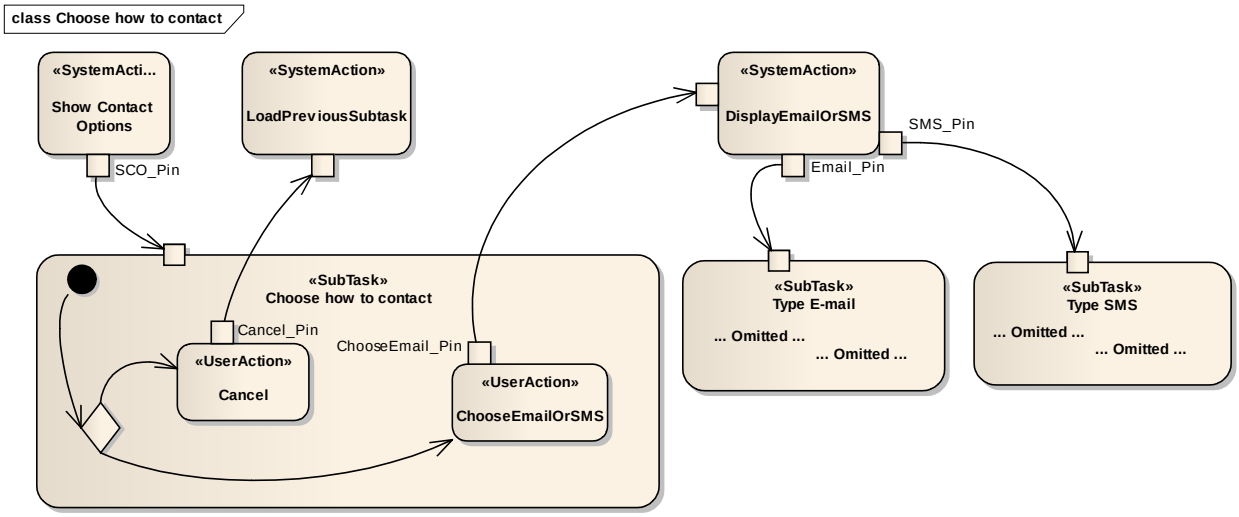

Figure 2: Example of Task-Action Model, showing the subtask "Choose how to contact".

3.2 Presentation model

The goal of the Presentation model (PM) in our approach is to describe the **structure** of the user interface. By structure we mean a set of widget types (i.e., buttons, icons, forms, etc.) and the specification of the containment hierarchy [2].

Rules for the construction of the PM are based on the meta-model for the presentation layer presented in [32] and have been included in the LPGM profile. The profile for the PM contains a hierarchy of stereotypes representing concrete and abstract interface objects. The hierarchy has a root in GElement which is a stereotype extending the UML meta-class and serves as a common parent for all inherited stereotypes that form a hierarchical tree. Tree leafs represent concrete interface objects (CIO) while tree nodes may represent abstract interface objects (AIO).

In the tree, three basic classes of UIOs are defined: containers, presentation units, and control units. These are the AIO and the parents of, e.g., Form, Label and ComboBox, respectively. Both AIOs and CIOs can be found in other approaches, such as [8, 17, 51] but we offer a richer set of UIOs: The UMLi approach (focused on both web and traditional UI) [51] provides three UIOs; the TEALLACH approach (focused on traditional UI) provides five UIOs [52], and the UWE approach (focused on web UI) provides ten UIOs [25]. If we need to eliminate a particular CIO, we can replace it by using the most appropriate AIO, which can be the nearest parent of the node in the hierarchical tree. Since all UIOs are defined in a UML profile, extension of to the set of UIOs is possible and easy.

We consider that the PM is a platform independent and reusable component that cannot include any information other than a structure. Specifying any of the geometrical aspects of the UIOs (location, width, height) or their appearance (color, font, alignment) a premature commitment to a specific look and feel. Therefore we decided to consider our PM as an artifact capturing the structure of the UI and nothing more. For us a structure means a set of UIOs and the logical relations among them. We have proposed in [29] a hypothesis, with which we can model the structure of a UI using hierarchical and neighborhood relations. In order to formalize a the description of the UI structure, we have formulated definitions that contribute to the definition of the UI structure.

Definition 1.
Let g denote a sorted couple (id, t) where the id is an indentifier and the t is a data type.

Definition 2.
Let G be a set containing all g elements.

Definition 3.
Let C be a set of containers: $C = \{(id,t) | (id,t) \in G \wedge t = container\}$

Definition 4.
Let VN denote a Vertical Neighborhood relation $VN \subseteq G^2$. This VN relation must satisfy Constraint 1.

Constraint 1.

$$\forall g_i, g_j : VN(g_i, g_j) \Rightarrow ((\neg\exists g_k : k \neq j \wedge VN(g_i, g_k)) \wedge (\neg\exists g_l : l \neq i \wedge VN(g_l, g_j))$$

If a g_i is in VN relation with a g_j then g_i cannot be in a VN relation with any other element.

Definition 5.

Let VN denote a Horizontal Neighborhood relation $HN \subseteq G^2$. This VN relation must satisfy Constraint 2.

Constraint 2.

$$\forall g_i, g_j : HN(g_i, g_j) \Rightarrow ((\neg\exists g_k : k \neq j \wedge HN(g_i, g_k)) \wedge (\neg\exists g_l : l \neq i \wedge HN(g_l, g_j))$$

If a g_i is in an HN relation with a g_j then this g_i cannot be in HN relation with any other element.

Relations of Horizontal Neighborhood and Vertical Neighborhood have additional constraint 3.

Constraint 3.

$$VN \cap HN = \varnothing$$

Definition 6.

Let H denote a relation $H = VN \cup HN$.

Definition 7.

Let ParentOf denote a relation $ParentOf \subseteq C \times G$. The ParentOf relation must satisfy the Constraints 4 and 5.

Constraint 4.

$$(c, g) \in ParentOf \Rightarrow \neg\exists d, d \neq c \wedge ParentOf(d, g)$$

No element g can have more then one parent c.

Constraint 5.

$$ParentOf(c, g) \Rightarrow \neg VN(c, g) \wedge \neg HN(c, g)$$

Neither c nor g can take part in any VH or HN relation.

We have expressed all defined relations in terms of UML and created stereotypes «ParentOf», «Neighbour», «H_Neighbour» and «V_Neighbour» as extensions of the UML Associaton meta-class. In our PM, we use the «ParentOf» stereotype to denote the first owned element of a container. The «Neighbour» stereotype denotes an ordered pair of elements. «H_Neighbour» and «V_Neighbour» are specializations of the «Neighbour» and correspond to the Horizontal and Vertical Neighborhood relations. We bind two UIOs by «H_Neighbour» or «V_Neighbour» when we want our model to represent these elements laid out horizontally or vertically, respectively, within a common container. With the use of these relations, we can model the containment hierarchy and the relations contributing to the UI layout.

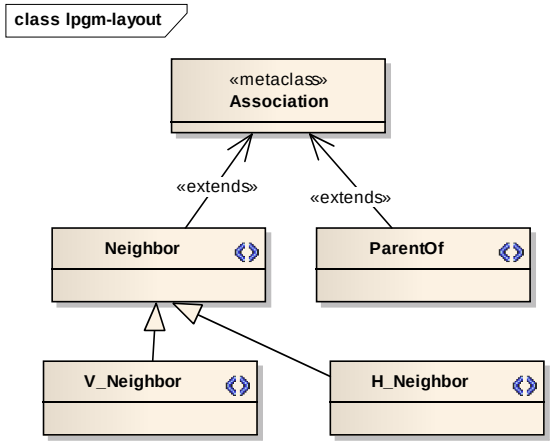

Figure 3: Stereotyped associations of the UML profile for the Presentation model.

3.3 Development environment

As we have mentioned, MB-UID is usually supported by a software tool that provides a graphical environment. Since we are focused on UML, we have explored several UML compliant modeling tools [30], [33] focusing on the level of their extension in order to support our modeling approach. These tools, which we have called **adaptive**, can be extended and adapted to specific purposes different from the original. In [33] we set forth the following requirements which must be satisfied in adaptive UML compliant modeling tools.

- The tool must support UML profiles and stereotypes as specified in [46]. We want to emphasize support for features enabling the application of alternative icons to stereotypes because appropriate icons can better convey the modeling domain thus and make the modeling more intuitive and clear.
- The tool must provide an API (Application Programming Interface) which enables access to the UML repository and manipulation of the data of the model.
- The API must provide a mechanism for establishing a channel of communication to show which action is being performed on the model data, e.g., element creation or model deletion. Such interactive observations enable keeping track of a development process and better controlling it.
- If there exists an adaptive modeling tool, we recommend implementing an environment that provides such functions as model transformations, generation of unique internal identifiers for new model elements and checking the names of elements according to naming conventions. The environment should also behave as a container for storing approach-compliant resources (transformation rules, type mapping, UML profiles, etc.).

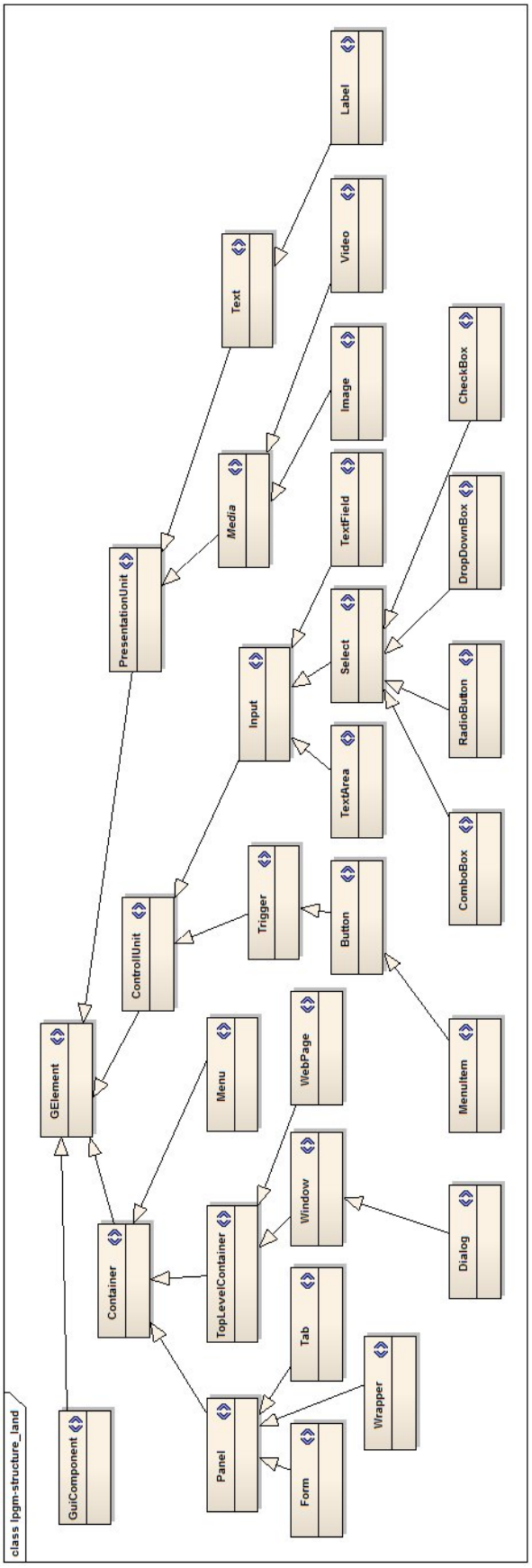

Figure 4: Stereotyped classes of the UML profile for the Presentation model.

4 Method application

The application of our approach consists of constructing models and generating source code for a particular platform. Models are built using the LPGM profile and are created without including any platform-specific information. The models created are considered as PIM models (Platform Independent Model) in the MDA. We will illustrate in this section how these models are constructed and utilized.

4.1 Model construction

Our approach to the process of modeling a user interface is based on the existing use-case model. After the functionality of the entire application has been specified, we identify all the goals of the given use-case for which the TAM is being constructed. Each goal is associated with a particular task, which is modeled as a «Task» activity. This activity is connected to the original use-case through «Realize» dependency. The task is broken down into «SubTasks» activities which contain «UserAction» actions. The first IOs are identified at this point: the PM is constructed in parallel, to enable it to immediately bind action pins «UserActionInfo» to UIs through «Supply» or «ActionTrigger» dependencies. A «SystemAction» action is constructed for each «SubTask», and a «SystemActionInfo» action pin is added to the action through «Present» dependency, the «SystemActionInfo» is associated with a container which is generated and displayed for the user in the next subtask. The container is produced by the «ActionProcessor» method, which represents physical implementation of the method used to generate the UI for the following task. The method is connected to the «SystemAction» through «Realize» dependency.

When the TAM is finished, the PM is completed by adding additional UIOs (images, icons, labels) to the IOs which were created during the construction of the TAM. All UIOs must be connected to their neighbors and parents through «ParentOf», «H_Neighbour», and «V_Neighbour» dependencies until the specification of the structure is finished. The PM is then done.

The system action "Show Contact Options" provides the “Controlls_Cont” container of the «Form» stereotype which is bound to the output action pin “SCO_Pin” through the «Present» association. This container is subsequently displayed in the “Choose how to contact” subtask, wherein the user can use two radio buttons “Email_RBtn” and “SMS_RBtn” in order to choose the way of communication or cancel the subtask. These two options are performed within “ChooseEmailOrSMS” and “Cancel” user actions, respectively. Thus the “ChooseEmailOrSMS” user action has the “CES_Pin” action pin, which is associated by means of two «Supply» associations with the “Email_RBtn” and “SMS_RBtn” radio buttons. Furthermore, the

"CES_Pin" has an «ActionTrigger» association connected with the "Next_Btn" button. The "Cancel" user actions its "Cancel_pin" action pin connected with the "Dismiss_Btn" button through «ActionTrigger» association. After using the "Next_Btn", the user action "DisplayEmailOrSMS" is executed and one the "Type E-mail" or "Type SMS" subtasks is displayed according to the user's choice.

4.2 Model transformations

The TAM and PM do not contain any information related to implementation since the target platform is not yet known. After the platform is specified we should transform current models in order to get new and richer models having a form optimal for the straightforward and effective generation of code. We have proposed several transformations which will be depicted in the next sections. The transformations are model-to-model and model-to-text and are applicable for both web and traditional UI.

4.2.1 Layout normalization of presentation model

After finishing the PM, we have the ideal model from the point of view of a developer. The developer need not focus on any implementation issues and the PM is thus created with respect its function and not technical issues. However, such a form of model is hard to interpret in the context of code generation. The problem is represented by so-called "**corner elements**". Corner elements are UIOs which take part in both V_Neighborhood and H_Neighborhood relations. Corner elements "Controlls_Cont", "Next_Btn", "Email_RBtn" and "SMS_RBtn" are seen in the fig. 5. Common containers of user interfaces can hold and arrange objects in only one direction, i.e., either horizontally or vertically: QT/C++ (HorizontalLayout and VerticalLayout), Swing/Java (BoxLayout.X_AXIS and BoxLayout.Y_AXIS), HTML/Web (div and span). Therefore we need to eliminate all corner elements in order to shift the model a bit towards an implementation form. We have proposed and implemented an algorithm [29] which breaks every corner element into one element and one new wrapping container. The element is later removed from the H_Neighborhood or V_Neighborhood relation and the relation is inherited by the new container (see fig. 6). This process is called "**layout normalization**" and after it is done all H_Neighborhood and V_Neighborhood relations must satisfy the constraints 6 and 7.

Constraint 6.

$$\forall g_i \in G : \exists g_j \in G : VN(g_i, g_j) \Rightarrow$$
$$\neg \exists g_k \in G : HN(g_i, g_k)$$

No element g_i can be in both VN and HN relations.

Constraint 7.

$$\forall g_i \in G : \exists g_j \in G : HN(g_i, g_j) \Rightarrow$$
$$\neg \exists g_k \in G : VN(g_i, g_k)$$

No element g_i can be in both VN and HN relations.

4.2.2 Model enrichment

The PM contains no additional information beside that information regarding the structure of the UI, so we need to add information through a transformation step which we call "model enrichment". Model enrichment is performed partly on the PIM level and causes the transition of the PIM to a PSM (Platform Specific Model), when the process of model enrichment begins to add platform-specific information. This enrichment is based on mapping "key - new information", where the key is a unique identifier of the model element being processed. New information can be added to the model in the form of tagged values, as has been demonstrated in [27].

Transformation at PIM level. Since no information related to appearance or content has been specified, we propose to add this through use of the tagged values `appearance`, `text` and `resource`. The tagged value `appearance` contains a link to the definition of appearance. It is not necessary to generate the `appearance` tagged value for all UIOs. It is enough to generate this for the top-level containers and distribute appearance information to their descendants at the run time (as we show in the next section). The tagged valued `text` contains either a text which will be displayed at the run time or a key referring to a resource that has a corresponding text value. The later approach enables flexible management of the content (e.g. localization) in future. This tagged value can be presented only by a UIO with the stereotypes Text and Label. The tagged value `resource` is generated for all types of Presentation units (i.e., Media, Image, etc.) and defines the location of an associated resource (e.g., multimedia file, image file).

Transformation at PSM level. Once the target platform has been chosen, we recommend enriching it immediately with additional, implementation-related information. This can typically be the data type for each UIO. For this purpose, we propose to set a `tptype` (target platform type) tagged value that refers to the fully qualified name of a data type for a UIO of a particular stereotype. Other tagged values can specify a namespace (C#) or package (Java) for top-level containers which are considered to be transformed into a class. We also propose to perform another enrichment which adds a new tagged value containing a text value that corresponds to an identifier suited to the target platform to prevent problems during source code compilation. The alternative name may be derived from the original one and can conform to a particular naming convention.

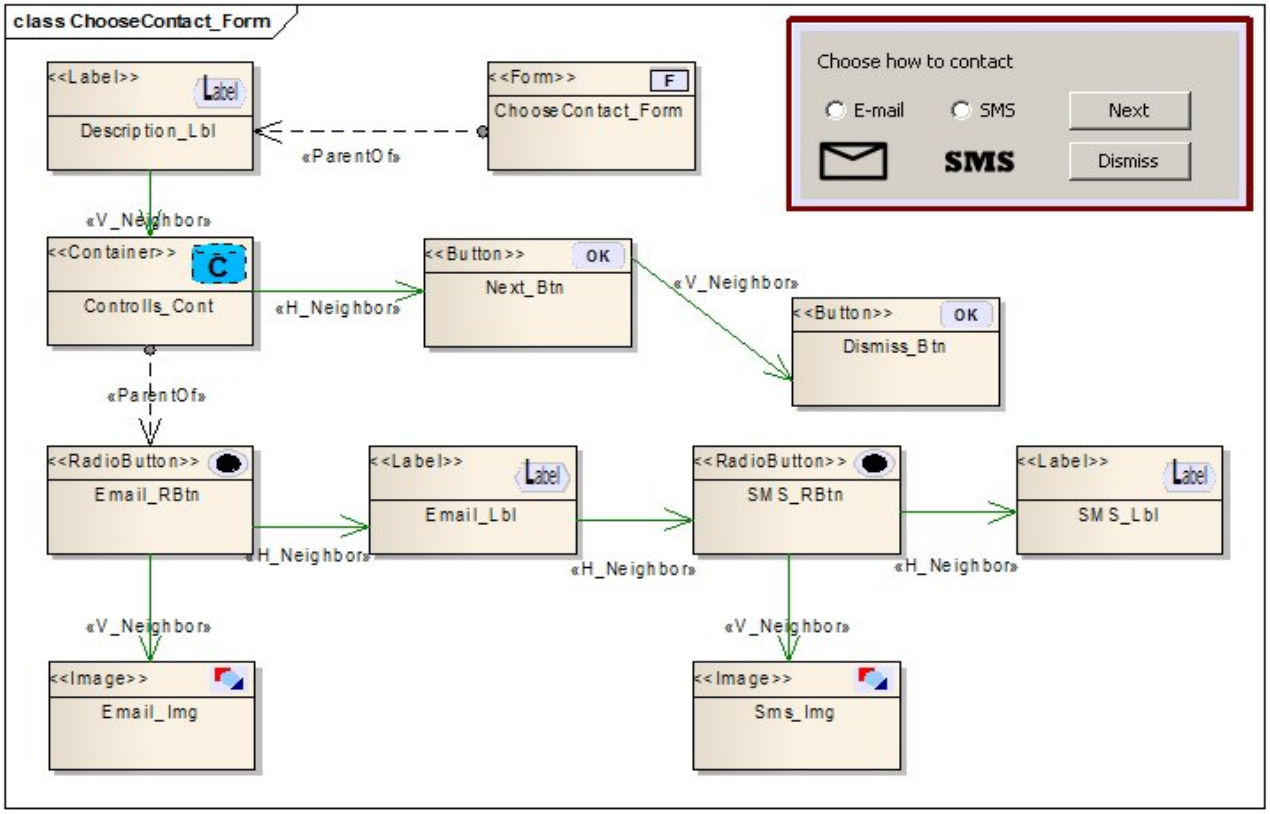

Figure 5: Depiction of the Container "ChooseContact_Form", an example of a PM. (A mock-up of it is shown in the upper right-hand corner.)

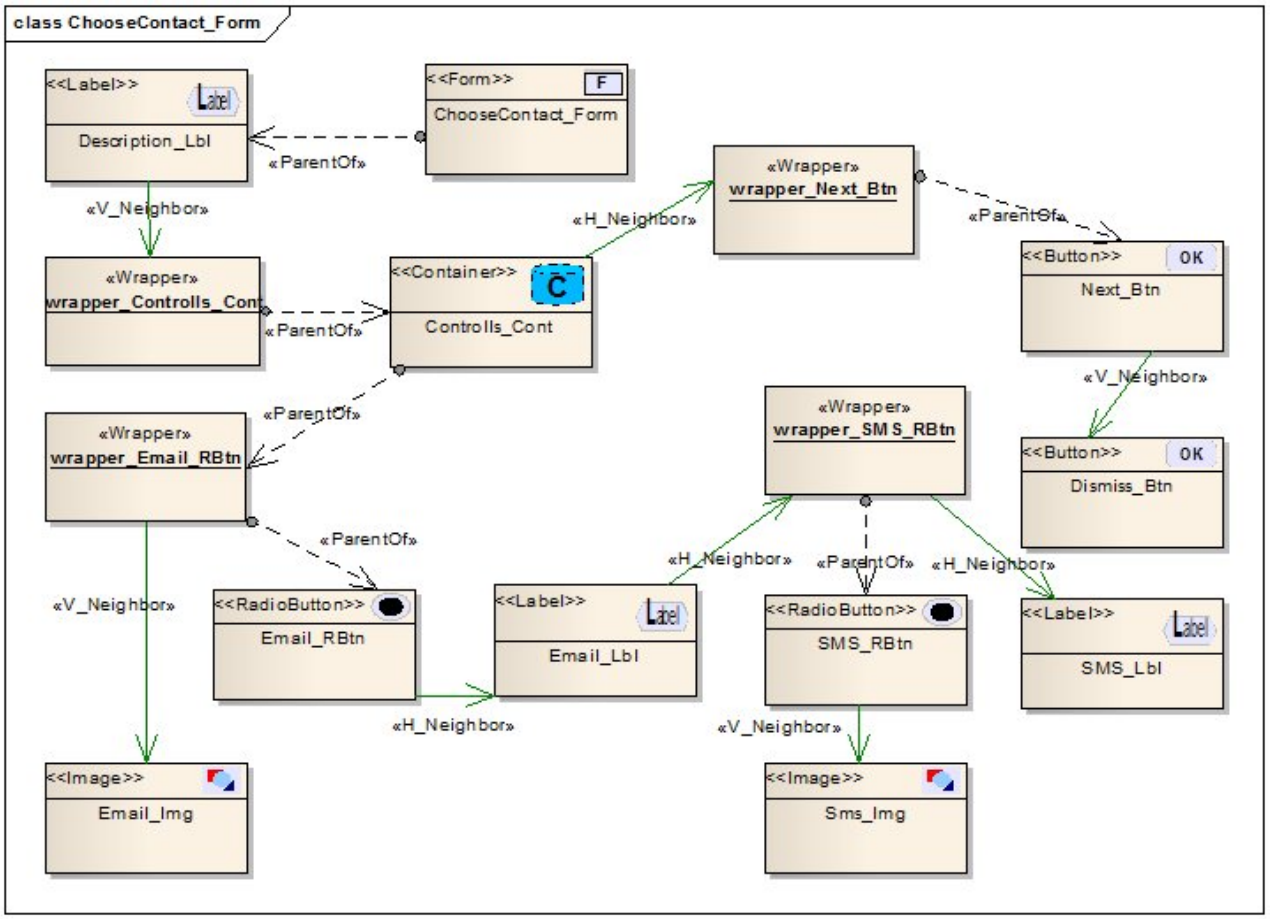

Figure 6: Normalized layout of the "ChooseContact_Form" container, which contains no corner-elements.

4.2.3 Source code generation

When the last model-model transformation has been completed, we can proceed to generate source code, remembering that one of our goals is to separate concerns as much as possible.

Based on the TAM, we propose to generate an XML file named Task-Action Descriptor, see fig. 10. This file provides information which can drive the application flow without the need to hard-code such information into the application logic or the presentation logic. By keeping track of both the last action triggered by the user (e.g., button press) and the TAD we can determine the corresponding action of the system and launch it dynamically.

We use the PM to generate source code for the user interface. Since we have not included any spatial information regarding the layout within the PM, we need to reconstruct this from the definition of the UI structure and place UIOs at the right positions within the top-level container. In traditional MB-UID, the UI layout is sometimes generated from a task model [35, 52, 5] or based on some grouping relations [24, 57, 52]. Some of these approaches use particular strategies that give a solution for automated placement. Techniques like the double-column strategy or right-bottom strategy [5], [56] provide good results under certain circumstances and only partially, so they cannot be employed widely without corrections [44]. The problem with these strategies comes from the endeavor to solve this issue complex and in their own hook. Therefore we decided to avoid generating source-code, including the command for the automated placement of UIOs. On the other hand, we propose to generate a UI layout with the use of containers which control the placement of UIOs on their own. This strategy can be applied in a variety of widget toolkits which support the concept of Layout Managers [18]. The great advantage of using layout managers is that they can adjust layout dynamically, e.g., during changes in the size of the screen.

5 Evaluation

We have already done some experimental evaluation of our method in the areas of traditional and web development. The first tests focused on generating a traditional UI for the Swing platform, where we employed our PM. The second test focused on web applications, particularly on the J2EE platform, where we employed both models with emphasis on the TAM. We used the reflection mechanism [45] intensively during this evaluation because our method is heavily dependent on it.

5.1 Development environment

In order to provide software support for our approach, we have implemented the software environment LPGM4EA, see fig. 7. We focused on contemporary UML-compliant modeling tools used in the commercial sphere because we wanted to explore the possibility that our approach could be adopted without forcing anybody to abandon a tool currently in use. After comparing the modeling tools Visual Paradigm, Enterprise Architect and Rational Rose, we have implemented our environment in the Enterprise Architect modeling tool. The EA is widely used in the community of software developers and provides some important features which put it into the class of adaptive modeling tools.

The LPGM4EA environment is written in the .NET, has its own presentation layer, and runs in its own window outside the EA graphical environment. The LPGM4EA is connected to the running instance of the EA through a bidirectional communication channel which is based on listener which propagates user actions performed in the LPGM4EA to and from the EA. The application layer of the LPGM4EA is also able to access the UML repository of the EA without the running instance of the EA. This offline access mode is also supported in Rational Rose, but it is not supported in Visual Paradigm. This lack of offline UML data processing precludes processing data non-interactively which can cause the development process to break down: there can be a lot of models in the UML repository, and thus it should be possible to process data automatically without user intervention, as a batch.

The LPGM4EA provides functionality which enables the running model-model and model-text transformations. These transformations are template-based [4] for both model-model and model-text transformations. It also watches the UML repository and manages newly added or deleted model elements. Thus we are able to decorate new elements with an `lpgmid` tagged value which holds our internal identifier, generated uniquely for PM and TAM elements and to provide some assistance in correctly naming model elements.

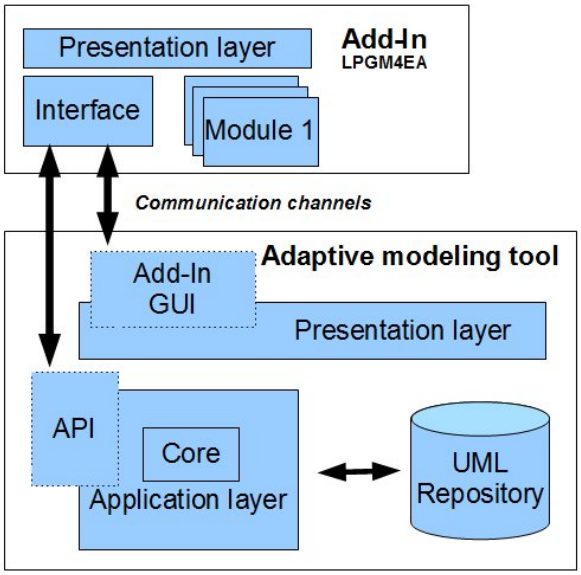

Figure 7: Scheme of the LPGM4EA modeling and development environment.

5.2 UI generation of Swing components and HTML tags

The application logic responsible for UI code generation is realized through the `IWireframe2CodeTransformation` interface, see fig. 8. A note on naming of our classes and interfaces: we think of the PM as a sort of (UML) wireframe and therefore we use the term "wireframe" within our implementation. The implementation of the interface is quite simple, and it is necessary to implement it on every platform we want to support. This interface is used by the interface `ITransformation2CodeController` which is able to read the normalized model. It drives the generation and calls methods placed in implementations of the `IWireframe2CodeTransformation`. At this moment, we have two implementations: `Wireframe2HtmlTransformationImpl` and `Wireframe2SwingTransformationImpl`. Each implementation has a template for the StringTemplate[1] library. Templates are designed with respect to the target platforms and provide parameterized generation of code.

5.2.1 Swing

User Interfaces for the Swing library were generated against top-level containers. A new java class was generated for each top-level container in the form of a file. This file (class) contains common sections such as a package name, an imports section, a class skeleton, attribute declarations and a constructor.

The import section is generated from the tagged value `tptype` of all the nodes and leafs in containment hierarchy of the top-level container. The class skeleton is generated for the top-level container and extends the platform type of the container (e.g. `class Foo extends JFrame`).

The section attribute declarations contains declaration expressions for all nodes and leafs (UIOs) in the containment hierarchy of the top-level container.

The constructor contains three blocks of commands. The first block contains commands for the initialization of all UIOs. The second block contains the commands responsible for building the containment hierarchy and setting the proper layout. We use BoxLayout with constants `BoxLayout.X_AXIS` and `BoxLayout.Y_AXIS` to lay out UIOs horizontally and vertically, respectively. The third block contains commands responsible for setting texts and resources for textual and multimedia UIOs. Figure 9 shows the part of the generated code that is relevant to figure 6.

The appearance of the components generated is set separately. It is achieved by creating a simple text file for each top-level container where in the appearance properties are specified for the particular UIO or group of objects. This definition is parsed and processed at run time. Then we proceed to set the appearance values of objects of the hierarchy of top-level containers. We get a reference to the top-level container object and traverse

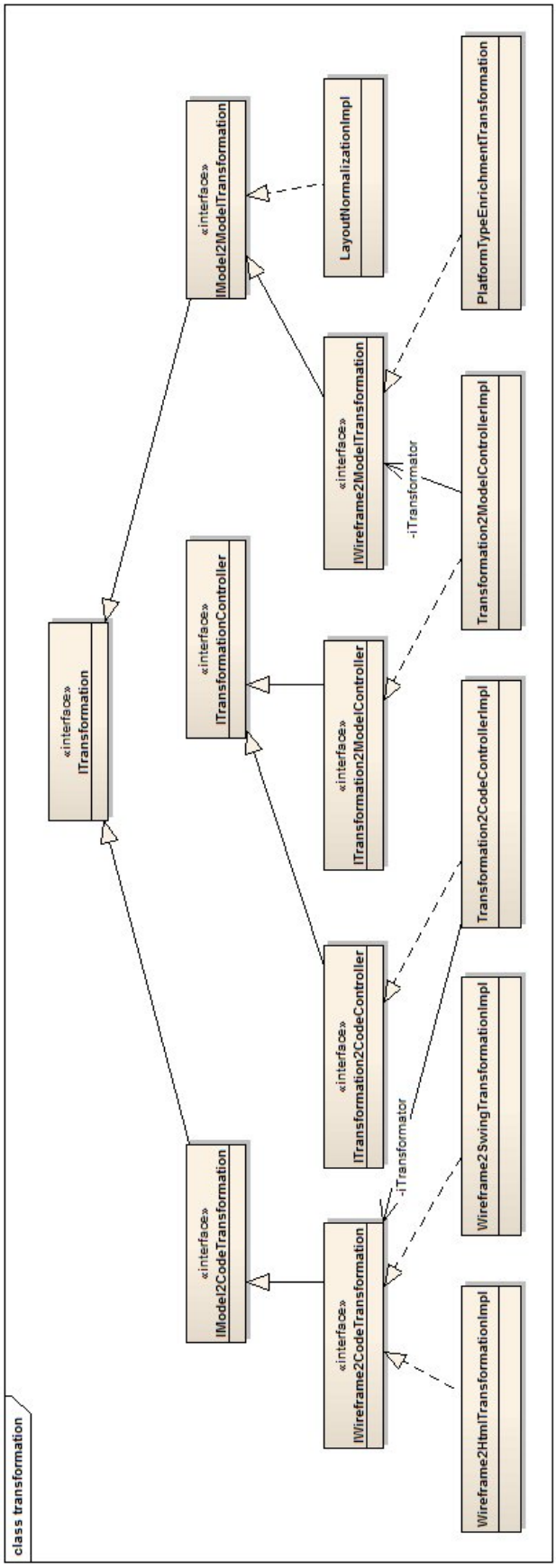

Figure 8: Depiction of the infrastructure of class and interface in the transformation process

[1] http://www.stringtemplate.org/

```
class ChooseContact_Form extends JPanel {
  ...
  private JPanel iControlls_Cont;
  private Container iWrapper_Email_RBtn;
  ..
  public ChooseContact_Form() {
    ...
    iControlls_Cont.setLayout(new BoxLayout(
    iControlls_Cont, BoxLayout.Y_AXIS ));
    iControlls_Cont.add(iWrapper_Email_RBtn);
    iControlls_Cont.add(iEmail_Img);
    iWrapper_Email_RBtn.setLayout(new BoxLayout(
      iWrapper_Email_RBtn, BoxLayout.X_AXIS ));
    iWrapper_Email_RBtn.add(iEmail_RBtn);
    iWrapper_Email_RBtn.add(iEmail_Lbl);
    ...
  }
}
```

Figure 9: Example of the code generated to set up the containment hierarchy for the diagram shown in figure 6.

the entire tree from the root to the bottom at the leafs level. Along the way we set all appearance properties to objects which are selected by a selector in the appearance-definition file. Each processed object is inspected via the reflection mechanism by checking for the existence of a method conforming to the name of the property and having an appropriate set of formal parameters. For instance, if there is a property "`font-style: courier, italic, 12`", we seek a method void `setFontStyle(String, String, int)`. If such a method exists, we perform its execution and supply the specified values.

In the generated file, the appearance is separated from the structure definition which makes the source code more modular and readable and easier to maintain. Furthermore, we claim that this strategy keeps up a unified appearance, thanks to the selector mechanism: `JLabel font: Font.BOLD` will apply the bold font style to all labels in the user interface. This prevents us from forgetting to set it, as we might if we were using the common manual approach.

5.2.2 HTML

During the PM evaluation, we also tested the generation of tags for HTML. This generation is easier thanks to the fact that HTML is a declarative language. In order to generate the structure of a UI in HTML, we used the `DIV` and `SPAN` tags to lay out UIOs vertically and horizontally, respectively. We were able to generate common HTML forms or menus for a content management system, where in the TAM was also utilized. The appearance was set in common way by using CSS (Cascade Style Sheet documents).

5.3 Dynamic flow control in web JavaEE applications

The TAM was tested during the development of a content management system on a JavaEE platform by using servlet and JSP technologies. We have designed a format for the XML file to hold information from the TAM. This document is called the Task-action descriptor.

The format of the document is self explanatory and corresponds to the TAM. Information in the document helps us to control the flow of applications. The descriptor contains records corresponding to the actions of a user and system which are bound via the `lpgmid` identifier. The utilization of the descriptor is performed according to the following scenario.

The user performs an action using a particular IO with associated lpgmid. The web browser generates an HTTP request and the `lpgmid` value is sent to the server as a parameter. The HTTP request is processed by a servlet, which extracts the lpgmid value and seeks the corresponding record in the TAD using `userAction`. The `userAction` found contains an attribute actionTrigger referencing a `systemAction`. The `systemAction` has a method name and the fully qualified name of the parent class. The method (`ActionProcessor`) is executed by the servlet via a reflection mechanism, and the `HttpServletRequest` is passed on as an argument. The method performs common steps such as extracting parameters, and calling application logic, and it generates an HTTP response (`HttpServletResponse`). The response contains a UI within the JSP specified as a view attribute. The UI is generated from the PM.

This way of processing an HTTP request replaces common techniques, where in long blocks of "`if-elseIf-...`" are used within the servlet code. Furthermore, if we need to change a flow order or UI generated for a particular subtask, we can do it manually by editing the TAD, without needing to compile compilation the servlet source code.

```
<systemActions>
  ...
  <systemAction="DisplayEmailOrSMS"
      userAction="375048">
    <nextSubtask="214781" parameters="Email_RBtn"
        view="/jsp/EmailForm.jsp"/>
    <nextSubtask="811626" parameters="SMS_RBtn"
        view="/jsp/SMSForm.jsp"/>
    </systemAction>
    ...
</systemActions>

<tasks>
  ...
  <task name="Notify co-workers" lpgmid="277193">
    ...
    <subtask name="Choose how to contact"
        lpgmid="724793" >
        <userAction name="ChooseEmailOrSMS"
            actionTrigger="375048"/>
        <userAction name="Cancel"
            actionTrigger="800064"/>
    </subtask>
    <subtask name="Type E-mail" lpgmid="214781"
        ...
  </task>
    ...
</tasks>
```

Figure 10: Depiction of the Task-action descriptor for the "Choose how to contact" subtask.

6 Future work

Our research and development within the LPGM approach is not finished. Our future activities will focus on more extensive utilization of the TAM with emphasis on generating the source code of event handlers in the scope of traditional UI or automating the extraction of parameters from an HTTP request and validating them. The models will also be used to generate technical and user documentation for the interface.

We also want to use the TAM to generate tasks for collaborative user interface agents [12]. We believe this is a good way of providing assistance to help users and support the user experience.

7 Conclusion

In this paper, we have introduced our approach for modeling and development of user interfaces. The approach can be classified as MB-UID since it is based on a set of models which are interpreted and used for transformations. The approach is suited to the field of traditional and web user interfaces. The constructed models can be used with a particular platform. The modeling approach focuses on task modeling in the context of a user and a system. Furthermore, it provides facilities to model the UI structure with the use of a PM. The TAM and PM have been formalized with the use of meta-model and algebraic formulas. These models can be processed automatically in order to perform a series of transformations resulting in the source code of the user interface for a particular platform. Processing of the models is supported by a software environment which provides assistance during the construction of the model and the generation of source code. Therefore the environment can be classified as MB-UIDE.

Our approach differs from other MB-UID approaches in several ways. Firstly, we use UML modeling notation in both our models, so they can be read and processed in other environments. This is a step towards interoperability and compatibility with industry standards. Secondly, our models can be considered as reusable components and can be used for the development of both web and traditional interfaces. It is not our goal to generate “ready to run applications” but just reasonable and useful fragments for the development of user interfaces.

We have demonstrated the utilization of our models with the support of our developing environment, which we have integrated into an adaptive modeling tool EA. The way we generate source code and integrate it into other source codes supports the separation of concerns. Such code is modular and easily maintained.

Acknowledgment

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

References

[1] Abouzahra, A.; Bézivin, J.; Fabro, M. D. D. & Jouault, F. (2005), A Practical Approach to Bridging Domain Specific Languages with UML profiles, in 'In Proceedings of the Best Practices for Model Driven Software Development at OOPSLA’05'.

[2] Batory, D.; Sarvela, J. N. & Rauschmayer, A. (2003), Scaling step-wise refinement, in 'ICSE '03: Proceedings of the 25th International Conference on Software Engineering', IEEE Computer Society, Washington, DC, USA, pp. 187--197.

[3] den Bergh, J. V. & Coninx, K. (2007), From Task to Dialog Model in the UML, in 'TAMODIA', pp. 98-111.

[4] Boas, G. E. (2004), 'Template Programming for Model-Driven Code Generation', http://www.softmetaware.com/oopsla2004/emdeboas.pdf.

[5] Bodart, F.; Hennebert, A.-M.; Leheureux, J.-M. & Vanderdonckt, J. (1994), Towards a dynamic strategy for computer-aided visual placement, in 'AVI '94: Proceedings of the workshop on Advanced visual interfaces', ACM, New York, NY, USA, pp. 78--87.

[6] Booch, G.; Rumbaugh, J. & Jacobson, I. (2005), Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series), Addison-Wesley Professional.

[7] Ceri, S.; Fraternali, P. & Bongio, A. (2000), 'Web Modeling Language (WebML): a modeling language for designing Web sites', Comput. Netw. 33(1-6), 137--157.

[8] Chesta, C.; Patern?, F. & Santoro, C. (2004), 'Methods and Tools for Designing and Developing Usable Multi-Platform Interactive Applications', PsychNology Journal 2(1), 123-139.

[9] Conallen, J. (2000), Building Web applications with UML, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[10] Constantine, L. L. & Lockwood, L. A. D. (2001), 'Structure and style in use cases for user interface design', 245--279.

[11] Domingues, A. L.; Bianchini, S. L.; Costa, M. L.; Ferrari, F. C. & Maldonado, J. C. (2007), eb application development methods: a comparison, in 'Workshop on Business Process Management'.

[12] Eisenstein, J. & Rich, C. (2002), Agents and GUIs from task models, in 'IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces', ACM, New York, NY, USA, pp. 47--54.

[13] Engels, G.; Heckel, R. & Sauer, S. (2000), UML -- A Universal Modeling Language?, in M. Nielsen & D. Simpson, ed.,'Proc. Application and Theory of Petri Nets 2000, 21st International Conference, ICATPN 2000, Aarhus, Denmark, June 2000.', Springer, , pp. 24--38.

[14] Fons, J.; Pelechano, V.; Albert, M. & Pastor, O. (2003), Development of Web Applications from

Web Enhanced Conceptual Schemas, in 'ER', pp. 232-245.
[15] Fowler, M. (2001), 'Separating User Interface Code', IEEE Software 18, 96-97.
[16] Frankel, D. (2002), Model Driven Architecture: Applying MDA to Enterprise Computing, John Wiley & Sons, Inc., New York, NY, USA.
[17] Griffiths, T.; McKirdy, J.; Paton, N. W.; Kennedy, J. B.; Cooper, R.; Barclay, P. J.; Goble, C. A.; Gray, P. D.; Smyth, M.; West, A. & Dinn, A. (1998), An Open-Model-Based Interface Development System: The Teallach Approach, in 'DSV-IS (2)', pp. 34-50.
[18] Haraty, M.; Nobarany, S.; DiPaola, S. & Fisher, B. (2009), AdWiL: adaptive windows layout manager, in 'CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems', ACM, New York, NY, USA, pp. 4177--4182.
[19] Jacobson, I.; Booch, G. & Rumbaugh, J. (1999), The unified software development process, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
[20] Karwaczyński, P. & Maciejewski, L. (2004), UML Proile for Analysis and Design of Jakarta Struts Framework Based Web Applications, in 'Proceedings of NWUML', pp. 185--196.
[21] Kirwan, B. & Ainsworth, L. K. (1992), A Guide to Task Analysis, Taylor & Francis.
[22] Knapp, A.; Koch, N.; Moser, F. & Zhang, G. (2003), ArgoUWE: A CASE Tool for Web Applications, in 'First Int. Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE 2003)'.
[23] Koch, N. (2001), 'Software Engineering for Adaptive Hypermedia Applications', PhD thesis, Ludwig-Maximilians-Universit?t M?nchen.
[24] Koch, N. & Kraus, A. (2002), The expressive Power of UML-based Web Engineering, in 'Proceedings Second Internatioanl Workshop on Web-Oriented Software Technology (IWWOST'02)'.
[25] Koch, N. & Mandel, L. (1999), 'Extending UML for Modeling Navigation and Presentation in Web Applications', online.
[26] Kong, X.; Liu, L. & Lowe, D. (2005), 'Separation of concerns: a web application architecture framework', Journal of Digital Information 6.
[27] Kozaczynski, W. & Thario, J. (2002), Transforming User Experience Models To Presentation Layer Implementations, in 'Second Workshop on Domain Specific Visual Languages'.
[28] Kruchten, P.; Ahlqvist, S. & Bylund, S. (2001), 'User interface design in the rational unified process', Object modeling and user interface design: designing interactive systems, 161--196.
[29] Kryštof, J. (2009): Formální popis rozložení prvků grafického uživatelského rozhraní, in 'The 11th nternational Conference MEKON'.
[30] Kryštof, J. & Chalupová, N. (2008), Prerekvizity pro novou koncepci modelování GUI v modelovacích nástrojích, in 'Objekty 2008', pp. 127-–136.
[31] Kryštof, J. & Motyčka, A. (2009), Extrakce scénářů do modelu úloh a akcí., in 'Objekty 2009'.
[32] Kryštof, J. & Motyčka, A. (2008), Metamodel for presentation layer, in 'Information Society', pp. 270--273.
[33] Kryštof, J. & Procházka, D. (2009), Rozšíření UML modelovacích nástrojů pro potřeby vývoje grafických uživatelských rozhraní, in 'Objekty 2009', pp. 264-–272.
[34] Lutteroth, C. (2008), Automated reverse engineering of hard-coded GUI layouts, in 'AUIC '08: Proceedings of the ninth conference on Australasian user interface', Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 65--73.
[35] Martínez-Ruiz, F. J.; Vanderdonckt, J. & Arteaga, J. M. (2009), Web User Interface Generation for Multiple Platforms, in 'Proceedings of the 7th International Workshop on Web-Oriented Software Technologies (IWWOST'2008) in conjunction with the 8th International Conference on Web Engineering (ICWE'2008)', pp. 63--68.
[36] Mišovič, M. & Turčínek, J. (2008), 'Teoretický přístup k tvorbě uživatelského rozhraní softwarových systémů', Acta Universitatis agriculturae et silviculturae Mendelianae Brunensis : Acta of Mendel University of agriculture and forestry Brno 6, 180--189.
[37] Myers, B. A. & Rosson, M. B. (1992), Survey on user interface programming, in 'CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems', ACM, New York, NY, USA, pp. 195--202.
[38] Myers, B.; Hudson, S. E. & Pausch, R. (2000), 'Past, present, and future of user interface software tools', ACM Trans. Comput.-Hum. Interact. 7(1), 3--28.
[39] Nierstrasz, O. & Achermann, F. (2000), Separation of Concerns through Unification of Concepts, in 'In ECOOP 2000 Workshop on Aspects & Dimensions of Concerns'.
[40] OMG (2010), 'UML Infrastructure specification', http://www.omg.org/spec/UML/2.1.2/.
[41] OMG (2007), 'XMI specification', http://www.omg.org/spec/XMI/2.1.1/.
[42] Paterno, F. (1999), Model-Based Design and Evaluation of Interactive Applications, Springer-Verlag, London, UK.
[43] Phillips, C. & Kemp, E. (2002), In support of user interface design in the rational unified process, in 'AUIC '02: Proceedings of the Third Australasian conference on User interfaces', Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 21--27.
[44] Puerta, A. & Eisenstein, J. (1999), Towards a general computational framework for model-based interface development systems, in 'IUI '99: Proceedings of the 4th international conference on

Intelligent user interfaces', ACM, New York, NY, USA, pp. 171--178.

[45] Rehak, M.; Tozicka, J.; Pěchouček, M.; Zelezny, F. & Rollo, M. (2005), An Abstract Architecture for Computational Re.ection in Multi-Agent Systems, in 'IAT '05: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology', IEEE Computer Society, Washington, DC, USA, pp. 128--131.

[46] Riesco, D.; Martellotto, P. & Montejano, G. (2003), 'Extension to UML using stereotypes', UML and the unified process, 273--293.

[47] Ryder, B. G.; Soffa, M. L. & Burnett, M. (2005), 'The impact of software engineering research on modern progamming languages', ACM Trans. Softw. Eng. Methodol. 14(4), 431--477.

[48] Schlungbaum, E. (1996), 'Model-based User Interface Software Tools - Current state of declarative models', Technical report, Graphics, Visualization and Usability Centre, Georgia Institute of Technology, GVU Tech Report.

[49] Schwabe, D.; de Almeida Pontes, R. & Moura, I. (1999), 'OOHDM-Web: an environment for implementation of hypermedia applications in the WWW', SIGWEB Newsl. 8(2), 18--34.

[50] Schwabe, D. & Gustavo, R. (1998), 'An object oriented approach to Web-based applications design', Theor. Pract. Object Syst. 4(4), 207--225.

[51] da Silva, P. & Paton, N. W. (2003), 'User Interface Modeling in UMLi', IEEE Softw. 20(4), 62--69.

[52] da Silva, P.; Paulo; Griffiths; Tony & Paton, N. W. (2000), Generating user interface code in a model based user interface development environment, in 'AVI '00: Proceedings of the working conference on Advanced visual interfaces', ACM, New York, NY, USA, pp. 155--160.

[53] da Silva, P. P. (2000), User Interface Declarative Models and Development Environments: A Survey, *in* 'DSV-IS', pp. 207-226.

[54] Tolvanen, J.-P. & Kelly, S. (2005), Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences, in 'Proceedings of the 9th International Conference on Software Product Lines, SPLC 2005', Springer, , pp. 198-209.

[55] Troyer, O. M. F. D. & Leune, C. J. (1998), 'WSDM: a user centered design method for Web sites', Comput. Netw. ISDN Syst. 30(1-7), 85--94.

[56] Vanderdonckt, J.; Ouedraogo, M. & Ygueitengar, B. (1994), A comparison of placement strategies for effective visual design, in 'HCI '94: Proceedings of the conference on People and computers IX', Cambridge University Press, New York, NY, USA, pp. 125--143.

[57] Viana, W. & Andrade, R. M. C. (2008), 'XMobile: A MB-UID environment for semi-automatic generation of adaptive applications for mobile devices', *J. Syst. Softw.* **81**(3), 382--394.

[58] WebRatioGroup (2010), 'WebRatio', online, http://www.webratio.com.

[59] Wirfs-Brock, R. (1993), 'Designing Scenarios: Making the Case for a Use Case Framework', *The Smalltalk Report* **3**(3).