INTEGRATING GUI PROTOTYPING INTO UML TOOLKIT

Darius Silingas[1,3], Saulius Pavalkis[1,2], Ruslanas Vitiutinas[1,3], Lina Nemuraite[2]

[1] No Magic Europe, Savanoriu av. 363, LT-49425 Kaunas, Lithuania, darius.silingas@nomagic.com, saulius.pavalkis@nomagic.com, ruslanas.vitiutinas@nomagic.com

[2] Kaunas University of Technology, Department of Information Systems, Studentu 50-308, LT-51368 Kaunas, Lithuania, lina.nemuraite@ktu.lt

[3] Vytautas Magnus University, Faculty of Informatics, Vileikos 8-409, LT-49425 Kaunas, Lithuania

Abstract. This paper introduces an extension of UML for modeling GUI prototypes. It presents the UML profile for GUI modeling, its implementation in MagicDraw, and its application to an experimental system. The profile contains stereotypes for the major GUI components that can be found in classic GUI libraries like Java Swing and several helper stereotypes and enumerations. While UML only allows defining an icon for a stereotype, the proper implementation of this profile requires rendering the symbols of the stereotyped elements as GUI components. This functionality was implemented as a plug-in to MagicDraw tool. The resulting solution enables storyboarding with GUI prototypes and linking their components with the other UML model elements like use cases, data class attributes, and states in GUI navigation state machines. These capabilities are demonstrated with examples from a test assessment system MagicTest, which is used for an experimental approval of linking the proposed profile with familiar software modeling artifacts.

Keywords: UML profile, GUI prototyping, storyboarding, model-driven development, MagicDraw.

1 Introduction

Unified Modeling Language (UML) is *de facto* standard in software modeling. Its importance has been amplified by the Model Driven Architecture (MDA) approach, which promotes modeling as the main development means. One of the UML strengths lies in its capability to visualize and relate multiple architectural views for a software system. However, UML lacks capabilities for modeling graphical user interface (GUI), which is an important software architecture view – the one that is most visible to the end user. In addition to its role in design, user interface prototyping is also used widely for gathering user requirements. A popular storyboarding technique focuses on capturing the user's actions using the system through a series of user interface snapshots providing representative examples of user inputs and system outputs. Currently, it is most common to capture such screen snapshots with hand-drawn sketches on the whiteboards or in drawings created in general purpose drawing tools like Microsoft Visio. These drawings can only be associated to UML model elements by external hyperlinks as they lack semantic structure and cannot be represented as composite model elements. Storing them separately from the model makes it difficult to connect finer-grained GUI components with model elements and perform traceability, change analysis, validation and other tasks that are nicely supported in modern modeling tools like MagicDraw. There have been multiple attempts to promote modeling GUI with UML, but such approach has been harshly criticized for being unnatural and overcomplicated. This critic is valid as UML 2 defines 248 meta-classes and 13 diagram types. Obviously, GUI modelers need just a small subset of that. They also want to use GUI-specific terminology and properties for model elements, and render the created GUI models in diagrams like real GUI components. The later requirement is critical for understanding the created GUI models and gathering end user feedback. In order to solve these issues, we propose to extend UML with a profile for GUI modeling with a special rendering, which emulates GUI look and feel, and develop a domain-specific modeling environment that makes it easy to model GUI prototypes and hides the complexity of UML from the GUI modelers while maintaining the possibility to integrate GUI models with the other model artifacts. In the rest of the paper we will present the GUI modeling profile, describe its implementation as MagicDraw plug-in, and provide illustrative examples of applying it for a case study system.

The rest of the paper is organized as follows: in section 2 the related work is reviewed, in section 3 the proposed GUI prototyping profile is presented, in section 4 an implementation of this profile and prototyping environment is described, in section 5 a demonstration of applying the proposed GUI prototyping profile to an experimental system MagicTest is presented, and section 6 summarizes results and indicates future work.

2 Related Works

Despite the fact that user interface modeling spans almost all phases of Unified Process there is no workflow dedicated for it. In practice, GUI design decisions are of creative nature rather than based on some theory. Furthermore, despite the huge amount of modeling concepts, UML is still not comprehensive enough to be applied for modeling user interface. Therefore, multiple extensions of UML have been proposed for that purpose, e. g. [4], [19]. However, some of these extensions are too abstract for capturing GUI details [4], or too complex for being practical [19] when developing GUI prototypes. Even if it is obvious that GUI prototyping is an important software development activity, there is still no standard profile for GUI development.

From the early days of using UML, it was understood that domain modeling and GUI design should occur simultaneously [3], [18], [25], [23], [24], [34]. Use cases are common means for capturing user requirements, e.g. [8]. Naturally, Martinez et al. [23] provide a methodology for validating requirements as early as possible by analyzing GUI combined with use cases; Hennicker and Koch [24] are focusing on precise steps for systematic design of the presentation; in [25], GUI components are obtained from use cases and activity diagrams.

Recently, several novel model-driven frameworks for GUI development have been proposed. Software developing communities have accumulated their experience in GUI design in the form of patterns. The most prominent collection of model-driven GUI design patterns together with the overall pattern-driven and model-based GUI framework is presented by Ahmed and Ashraf in [1] where different kinds of patterns are used as modules for establishing task, dialog, presentation and layout models. Authors present XML/XUL [37] based tool for selecting, adapting and applying patterns to task models for generating GUI prototype, evaluating it and generating the final GUI for Windows XP platform. Inesta et al. [12] propose an adaptation of User Interface Markup Language (UIML) by defining a data model, services model, and a navigation model that allows data communication from one GUI to another. The obtained user interfaces together with Web Services can represent complete applications instead of just being prototypes. User GUI representation and code generation techniques are needed alongside existing model driven methodologies, e. g. [33]. Kapitsaki et al. [14] propose a presentation profile for modeling the GUI presentation of Web applications, the presentation flow and the application navigation properties. It is similar to the profile used by the AndroMDA code generator and uses UML State Machines for modeling distinct states of application objects and dependencies between these states and is thus suitable for service flow modeling. Funk et al. [7] propose to integrate observation functionality into model-driven GUI development process for collecting user data and testing user interface design. In [27], the modeling and animation of user interface to smart devices is proposed for evaluating GUI usability.

As one can conclude from these exponents of existing research, the total requirements for modeling GUI include the capabilities of representing, evaluating and even monitoring the features of GUI, integrated with the overall software product and its development lifecycle while graphical representation namely remains on rather abstract level. A generalization and further adaptation of existing approaches is needed [17]. Apparently, no standard way for GUI modeling exists suitable for usage in universal CASE tools and simultaneously capable for sufficiently representing GUI details.

Graphical view specific user interface details usually exist in integrated development environments that lack aforementioned common capabilities for model-driven GUI design. Consequently, the purpose of our work is to integrate the GUI prototyping functionality into a general purpose UML CASE toolkit equipping this target functionality with high fidelity, interactivity and exhaustive specification having in mind the further possibilities to relating a GUI design with the development process and generating its implementation.GUI prototyping can be integrated into UML toolkit in a few ways: by integrating with 3rd party tool, by hard-coding shapes, or by creating Domain Specific Language (DSL) [31] for representing such shapes. Each solution has its benefits and shortcomings. Different works exist in areas related with each of these solutions.

The first one is the 3rd party component integration. The comparison of GUI prototyping solutions [13], [9] showed that there are not many Java Swing stand-alone tools that could provide handy prototyping solution. The two most potential candidates are Ribs [28] and Petra [22]. There are also tools dedicated for Java GUI creation, however, it is too cumbersome to quickly create prototype with such tools. Ribs is good for prototyping as it provides an easy creation of layout, however, some objects as tables and tree elements are not customizable. Petra suggests easy prototyping for Web-oriented design, however, it comes as Eclipse plug-in. Integrating them into UML tool would allow having realistic components for high-middle fidelity prototyping or low fidelity (wireframe) prototyping according to prototyping techniques described in [32], [38]. However, the 3rd party solution would be hardly extendable or adjustable – low compatibility with the toolkit, hard fixing of bugs.

The second possibility is based on hard-coding of simple shapes for low fidelity prototyping with DSL consisting of scalable and static icons, text and layout for them as in [6]. Solution is implemented using existing UML tool extension engine. It allows defining images and label positions on shape, initial shape size, etc. However, due to engine limitations, it has no advanced elements such as trees, existing elements lack usability.

In order to have the high fidelity of GUI and its flexible, standard integration into UML toolkit, DSL with hard-coded, advanced shape support is required. Similar solutions are implemented in [36], [29]. However, solution [36] requires too much detail to be specified, not resulting in better GUI communication. Solution [29] has quite low usability, no editing on screen, no look and feel support, any good integration with other UML-based domains. Project reload is required for changes to be refreshed across UML and GUI toolkits.

In order to fill in this gap in universal CASE tools, we have created a profile for GUI prototyping allowing designing high fidelity, interactive GUI prototypes with detailed specifications. In contrast to many approaches, the profile provides a platform-independent notation familiar to graphics designers. Our GUI prototyping profile is based on UML component and component realization metaclasses that allow reuse of GUI components in multiple screens whereas most of existing solutions are based on class or object [11], [25], or composite structure [23], [15], [38] modeling, so they are limited to a single owner per element. Our solution is agile, method independent, suitable for major software development approaches. Possibilities for connecting GUI prototyping with different software modeling stages are presented in illustrative examples in the section 5.

3 Defining UML Profile for GUI Prototyping

Based on OMG meta model architecture, a software architect needs to prepare the model of *System Architecture* at M1 meta level, which is constructed usign *UML* as a metamodel (modeling language) at M2 meta level. A significant part of *System Architecture* is *GUI Prototype*(s) that cannot be represented nicely with UML itself, and thus need to be based on *GUI Modeling Profile* extending UML, see Figure 1. Designing this profile and constructing the modeling environment for using it productively is the problem domain of this paper.

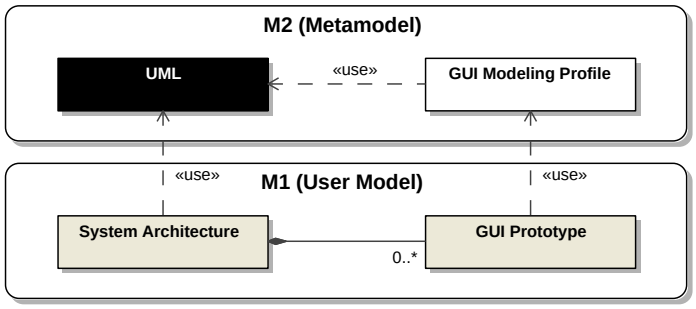

Figure 1. GUI prototype modeling problem domain

GUI Prototyping Profile extends UML 2 in order to support stereotypes for user interface prototype modeling. The profile is customized for usage in MagicDraw UML tool. This gives user capabilities to model GUI prototypes and render them as actual GUI components. It also enables to integrate GUI prototypes as a particular view of the architecture model and relate them to the other architecture model elements using standard UML relationships and tool-specific hyperlinks.

Profile is oriented to software applications GUI prototyping. It supports a minimal set of elements and their properties that are used in most applications for GUI prototyping needs. It is independent from a specific GUI look and feel and does not cover details necessary for actual GUI development and layout of component.

All user-defined GUI prototype components will be represented by underlying UML model elements with stereotypes from GUI Prototyping profile. GUI element-specific properties are taken from corresponding stereotype tags. Profile defines stereotypes for simple and more advanced customizable GUI, e.g. button, label, tabbed pane, table, tree, etc. All stereotypes defined by the profile are grouped into packages based on GUI element type – containers, buttons, text, and other, see Figure 2.

GUI Prototyping profile is based on UML 2 components, classes, and components realizations [20]. Stereotypes for composite GUI elements, e.g. *Frame*, *Panel*, *GroupBox*, extend *Component* metaclass. Stereotypes for atomic GUI components, e.g. *TextField*, *ScrollBar*, *Node*, extend Class metaclass. A standard UML *ComponentRealization* relationship is used to represent composite GUI element realization by its component GUI elements. When modeling GUI prototypes, *ComponentRealization* relationships should be created automatically. Using component realizations enables to reuse the same GUI component in multiple composite GUI components. Stereotypes hierarchy in the profile is designed to enable reuse of the tag definitions in multiple GUI element types. General properties reused in multiple GUI components are grouped to abstract Stereotypes of GUI Prototyping profile. Figure 3 presents *TitledComponent* hierarchy design as an example.

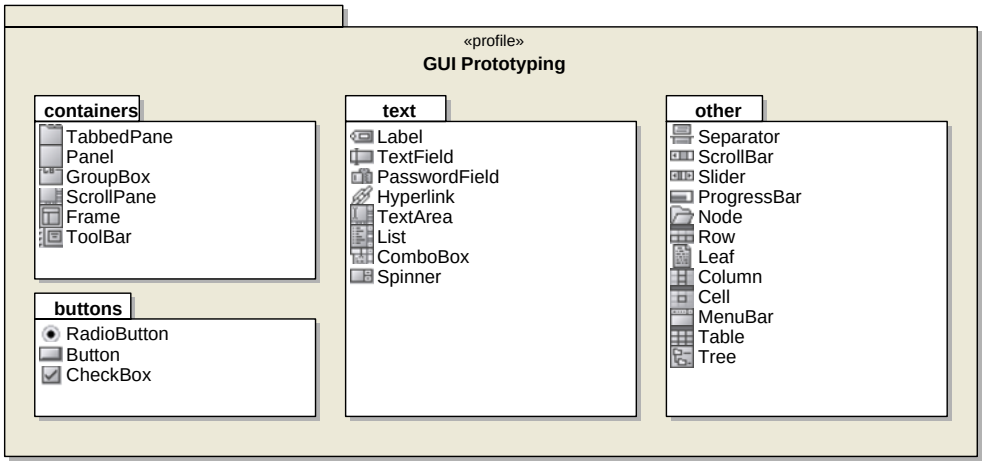

Figure 2. GUI Prototyping profile structure

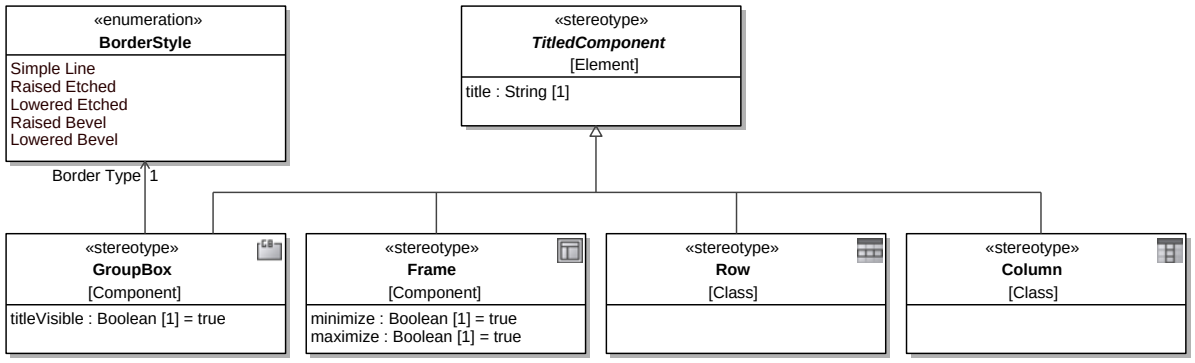

Figure 3. GUI stereotype design example (excerpt)

4 Implementing GUI Prototyping Support in MagicDraw UML

Implementation of GUI prototyping using stereotyped standard UML elements in modeling tools requires overriding default graphical model elements representation by typical GUI component view. It also requires updating graphical GUI component view due to model element or diagram symbol properties changes. Additionally, it is recommended to define a custom diagram for GUI prototyping. MagicDraw enables implementing these requirements using DSL engine (for creating custom GUI modeling diagram) and Open API (for developing a plug-in for custom rendering of GUI component symbols).

MagicDraw clearly separates model element data specification and visualization of model elements as symbols in diagrams according to the Reference Model Pattern [10]. MagicDraw model elements data properties are encapsulated in classes implementing OMG UML 2.2 metamodel following the principles of Java Metadata Interface [5]. MagicDraw Open API provides ability to override standard UML element symbols representation in diagrams using custom renderers. Renderer is a common pattern used for separating visual components from their drawing algorithms, allowing dynamic determination of visual appearances [10]. MagicDraw Open API defines *ShapeRenderer* class and *PresentationElementRendererProvider* interface as a *Renderer* pattern for custom symbol rendering implementation. Developers may define custom symbol rendering by overriding *ShapeRenderer* operation *draw* and implementing *PresentationElementRendererProvider* interface for specific symbol drawing. Interface *PresentationElementRendererProvider* implementation should be registered in MagicDraw using *PresentationElementRendererManager* operation *addProvider*. A universal custom renderer and renderer provider has been implemented for all GUI Prototyping profile stereotypes, see Figure 4.

Java Swing library was chosen as a provider for GUI component symbol rendering. GUI component symbol drawing is delegated to the corresponding Java Swing component *paint* operation. Determining which component should be drawn is implemented using a modified *Visitor* pattern [21]. *JComponentRenderer* uses *JComponentHandlerAcceptor* for performing appropriate logic using *JComponentHandler* interface realizations for specific GUI symbols needs.

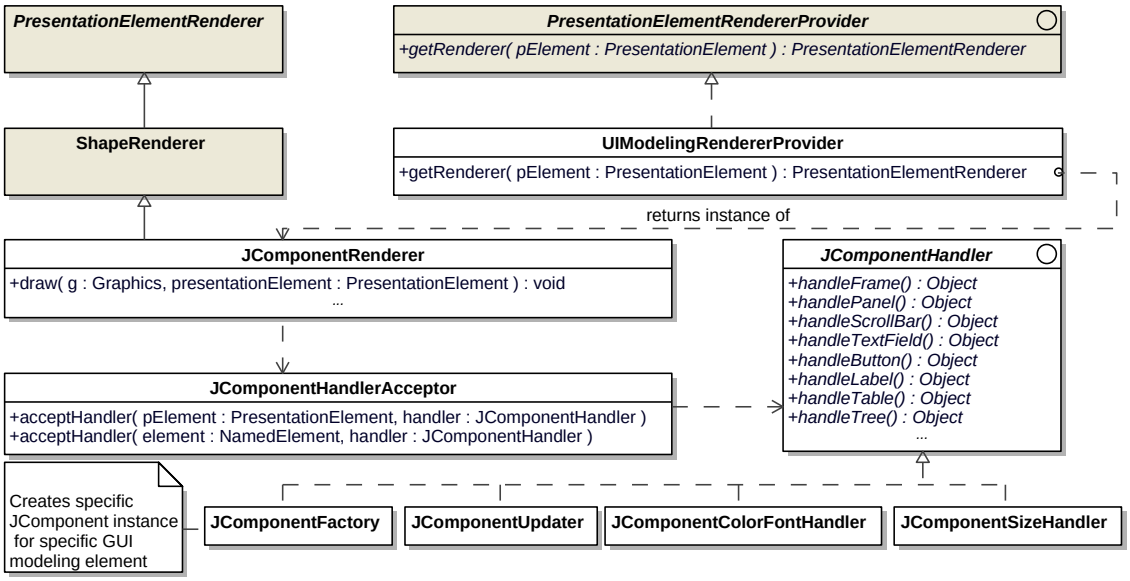

Figure 4. Major MagicDraw plug-in classes for custom rendering of GUI component symbols

GUI symbols rendering depends on the model elements properties, thus it must be updated whenever elements properties change. MagicDraw provides model changes notification mechanism using *Listener* pattern [16]. Listeners for handling relevant updates of GUI component model element data have been implemented.

In addition to the plug-in for custom rendering, a custom GUI Prototyping diagram was created based on the GUI Prototyping profile using a typical workflow for creating domain-specific modeling environment as presented in [31]. A snapshot of this environment is presented in Figure 5.

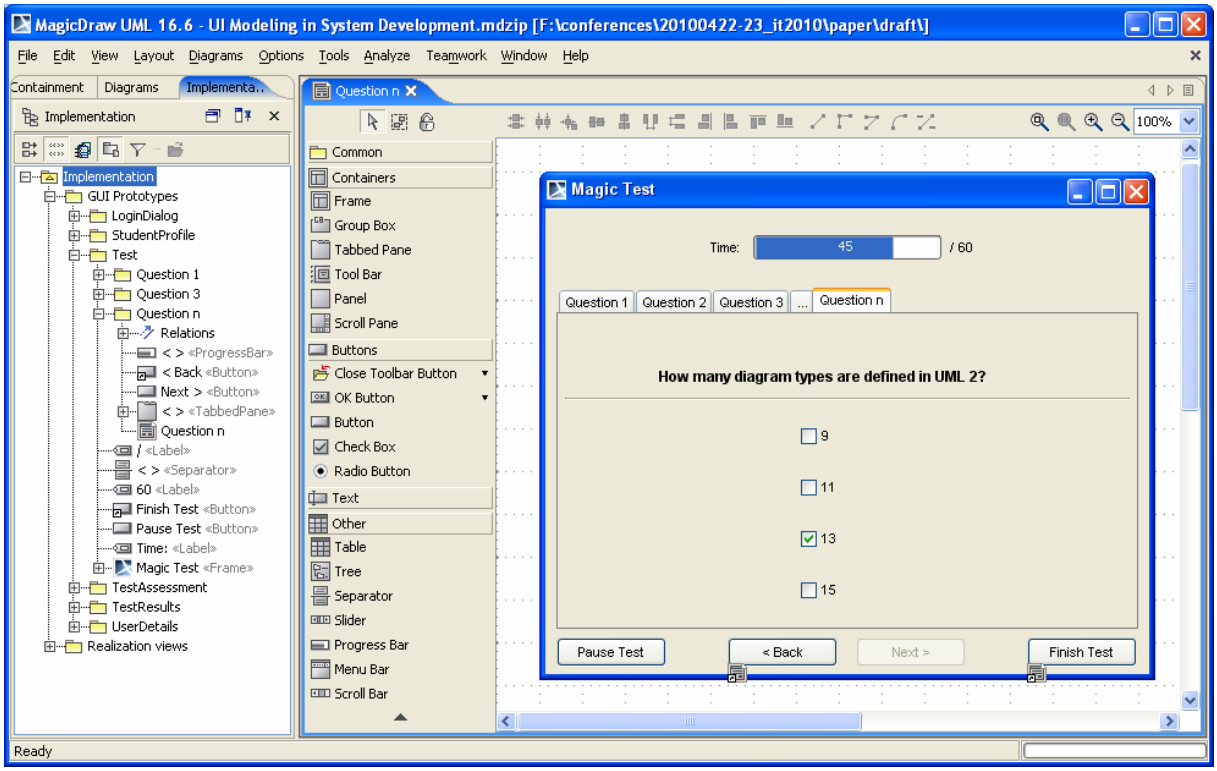

Figure 5. MagicDraw environment customized for modeling GUI prototypes

Due to UML model based GUI prototyping implementation, the standard MagicDraw features and UML constraints apply for GUI models, e.g. nesting, editing on the diagram, symbol properties support, model data specification dialogs, validation, report generation, export as image, etc.

5 Applying GUI Prototyping to Experimental System

GUI prototyping can be used in various project development stages. We will discuss the most common uses of GUI prototyping in the context of modeling, and give several examples from the modeling of an experimental system MagicTest [30], which automates student evaluation by automated test assessments.

Starting with requirements analysis, one typically defines use cases for the system under design. The end users will invoke the uses cases through GUI, thus a better understand of end user requirements can be achieved by building GUI prototypes driven by use cases. For storing this information in the model, we recommend to relate one or more GUI prototype model elements to each use case model element using a standard UML *Realization* relationship, see Figure 6. It is also advised to keep uses cases and GUI prototypes in separate packages as the former one is a part of an abstract platform-independent model (PIM), and the latter one is a part of a more concrete platform-specific model (PSM). Transition from abstract presentation model to more concrete one by refinement is analyzed in [23]. In MagicDraw, a modeler can assign hyperlinks to GUI prototype buttons or any other model element. This enables navigation through prototypes for simulating an abstract use case realization. A behavior of complex use case is typically specified using activity models. GUI prototypes can be hyperlinked on activity actions for illustrating GUI invocation in a particular use case scenario. Activity and GUI Elements relation is analyzed in [2], [23], [34]. Theoretically, it is possible to implement a model to text transformation, which transforms the model into executable prototype realization by generating UIML [35], XUL [37] or another user interface specification language scripts. For completeness of use case coverage in GUI prototypes, it is easy to set up an automated relationship matrix, which is a standard feature of modern UML tools like MagicDraw. The analysis of use case and GUI prototype relationships is very common in practice and research [2], [3], [18].

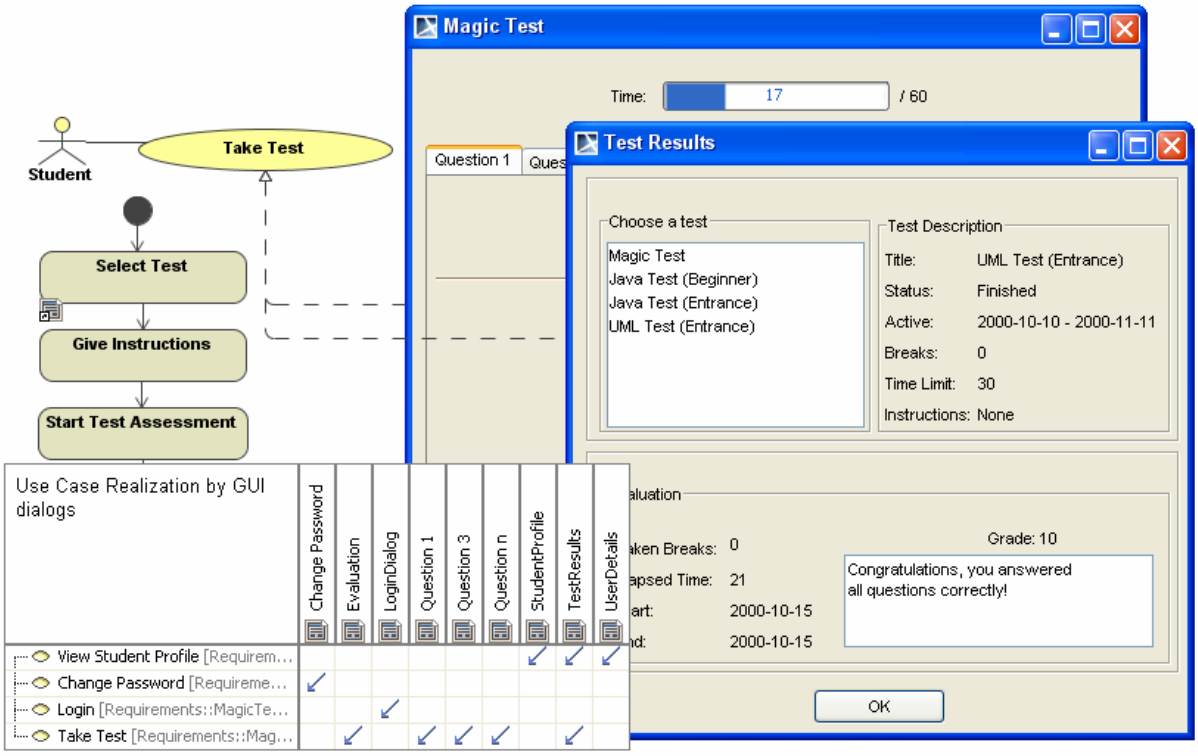

Figure 6. Fragments of use case model, screen prototypes, and relationship matrix

One of the first tasks for shifting from requirements analysis to design is identifying the major components in all system layers. Robustness analysis is a common technique for identifying user interface components (called *boundaries*), services (called *controls*), and data structures (called *entities*), which is supported in most UML tools [30]. A modeler can obviously link each boundary element to one or more GUI prototypes. However, this gives just a fragmented view of each GUI element as a separate component. For the overall understanding of user interface navigation it is recommend to define a state machine representing transitioning between screens possibilities for a particular actor [30]. In such a state machine, each state represents using a particular GUI component, which makes it a natural candidate for assigning hyperlinks for navigating to one or more GUI prototypes.

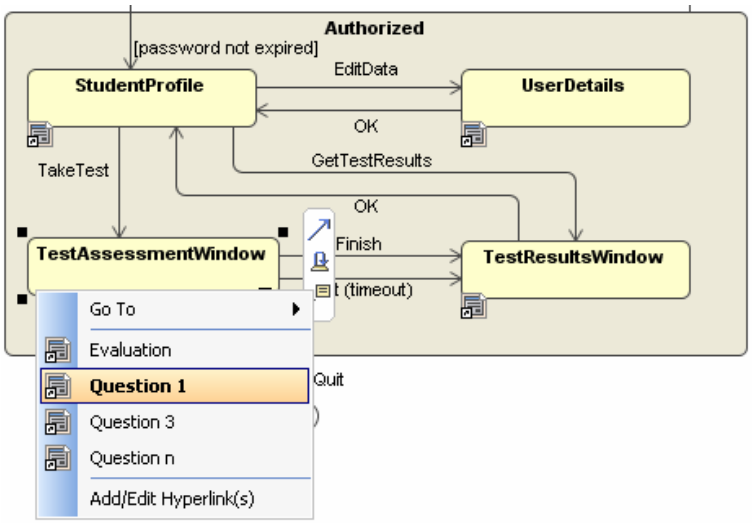

Figure 7. GUI navigation schema with hyperlinks to multiple GUI prototypes

It is necessary to make sure that the information that needs to be shown in GUI components is available in data structure. When a detailed data structure is modeled, it is very useful to relate fields of a specific GUI prototype with properties of data classes, see Figure 8. This may help to identify inconsistencies between user interface and data layers and change one or the other accordingly. These relationships are later very useful for traceability and change impact analysis purposes.

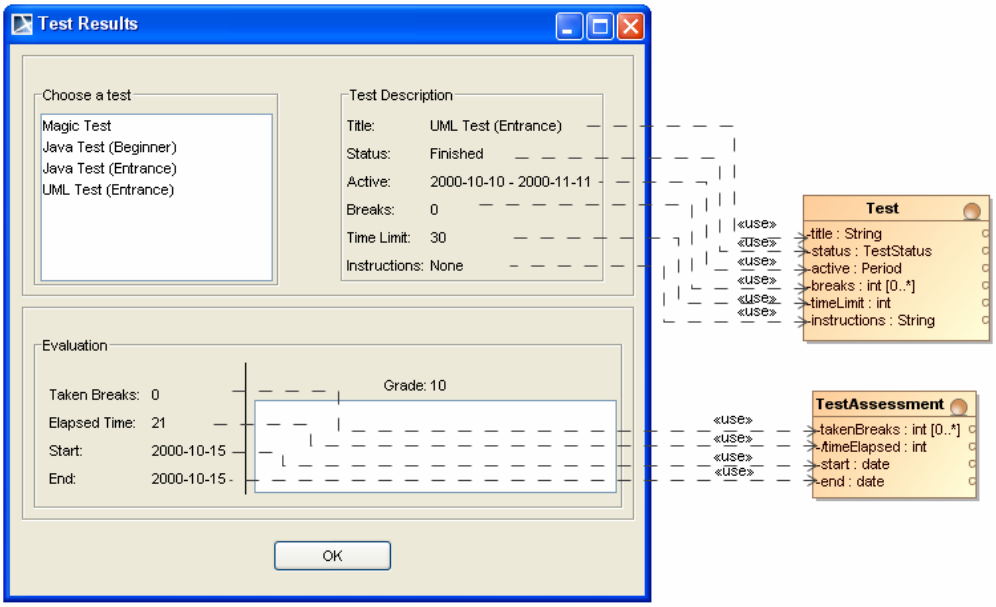

Figure 8. Mapping of user interface component fields to data class properties

Table 1 consolidates the potential uses of GUI prototyping in a typical modeling project following the modeling process presented in [30].

Table 2. Applying GUI prototyping in a modeling project

Project stage	Standard UML model elements	Usage of GUI prototyping	Relationships to standard UML model elements
Requirements	Use cases	GUI prototypes are modeled to better understand the requirements based on use cases	Realization relationship from GUI prototype to use case
Requirements	Use Case activities	Linking GUI prototypes to actions enables easy simulation of use case scenarios	Navigable hyperlink from action to GUI prototype
Architectural design	Robustness analysis model	Linking GUI prototypes to boundary classes enables easy visualization of their	Navigable hyperlink from boundary class to GUI

Project stage	Standard UML model elements	Usage of GUI prototyping	Relationships to standard UML model elements
		structure	prototype
Detailed design	State Machine for GUI navigation	Linking GUI prototypes to states enables easy simulation of navigation scenarios	Hyperlink represented on diagram.
Detailed design	Data classes	Linking GUI prototype fields to data class properties assures consistency between GUI and data layers and enables traceability	Usage relationship from GUI component field to data class property

By connecting a number of GUI prototypes, linking them via hyperlinks on buttons, and transforming the model into browsable report [31], it is possible to present a story of using the application. Such GUI prototyping approach is called *story boarding* [11], [26], [38] and is quite popular in practice.

6 Summary and Future Work

We have defined a UML profile, which contains stereotypes for modeling major GUI elements, and implemented MagicDraw plug-in, which renders the symbols of the stereotyped GUI model elements as Java Swing components. The plug-in also includes domain-specific customizations for the stereotypes and a custom GUI Modeling diagram that make it easier and more productive to apply the profile. We have validated the suitability of this extension by modeling GUI interface for a case study system MagicTest and connecting it to the other model elements, which integrates the GUI prototypes into the overall system architecture model. The plug-in has been released as a part of a commercial MagicDraw product. It has already been taken in use by many MagicDraw tool users who appreciated it and provided multiple suggestions for improvements.

In the future, we are planning to re-factor the profile for a better integration with the composite structure concepts, and enable switching between different look and feel libraries independent from MagicDraw look and feel that would enable the emulation of sketching and webpage design. We are also considering:

- Conversion to working realization, by generating XUL [37] or other user interface description language and aligning GUI Prototyping profile capabilities with XUL;
- Improving usability of relating various model elements with GUI prototype elements;
- Defining validation suites [29] for completeness and style conformance of GUI prototype models;
- Supporting internationalization of GUI prototype textual properties;
- Providing predefined but extendible libraries of reusable GUI prototype components;
- Defining open API for enabling MagicDraw users to customize and extend GUI prototyping.

The popularity of the GUI prototyping plug-in and multiple directions for improvements prove the importance of having GUI prototyping as a component in the UML toolkit. We will continue our scientific and practical research and development activities targeted at making it even more useful for the modelers.

7 Acknowledgements

We would like to thank Dennis Pingel, who has built a prototype of the MagicDraw plug-in for GUI prototyping based on the provided requirements during his internship at No Magic Europe, and Mindaugas Genutis, who supervised Dennis and later evolved the prototype into a high-quality product.

References

[1] **Ahmed, S., Ashraf, G.** Model-based user interface engineering with design patterns. *Journal of Systems and Software, Vol. 80(8),* August 2007, 1408–1422.

[2] **Almendros-Jimenez J., Iribarne L.** Designing GUI components from UML Use Cases. *In Proc. 12th Int. Conf. and Workshop on the Engineering of Computer Based Systems*, 2005, 210–217.

[3] **Blankenhorn K., Jeckle, M.** A UML Profile for GUI Layout. *In Object-Oriented and Internet-Based Technologies, LNCS, Vol. 3263*, 2004, 110–121.

[4] **Conallen, J.** Building Web Applications with UML. *Pearson education*, 2003.

[5] **Dirckze R.** Java™ Metadata Interface(JMI) Specification. *Java Community Process, Version 1.0, 2002*

[6] Enterprise architect: UI Modeling extention, http://www.sparxsystems.com.

[7] **Funk, M., Hoyer, P., Lin, S.** Model-driven Instrumentation of Graphical User Interfaces. *In Second International Conferences on Advances in Computer-Human Interactions*, 2009, 19–25.

[8] **Gudas, S., Lopata, A**. Meta-model based development of use case model for business function. *Information Technology and Control, Vol. 36(3)*, 2007, 302–309.

[9] GUI prototyping tools, http://c2.com/cgi/wiki?GuiPrototypingTools.

[10] **Heer J., Agrawala, M.** Software Design Patterns for Information Visualization. *IEEE Transactions on Visualization and Computer Graphics 12 (5):* 853. 2006

[11] **Hennicker, R., Koch, N.** Modeling the User Interface of Web Applications with UML. *Workshop of the pUML-Group held together with the «UML»2001 on Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists*, October 01, 2001, 158–172.

[12] **Inesta, L., Aquino, N., Sánchez, J.** Framework and authoring tool for an extension of the UIML language. *In Advances in Engineering Software: Designing, modelling and implementing interactive systems. Vol. 40 (12)*, December 2009, 1287–1296.

[13] Java GUI Builders: Java Swing GUI builders review, http://www.fullspan.com/articles/java-guibuilders. html.

[14] **Kapitsaki, G. M., Kateros, D. A., Prezerakos, G. N., Venieris, I. S.** Model-driven development of composite context-aware web applications. *Information and Software Technology, Vol. 51*, 2009, 1244–1260.

[15] **Koch, N., Baumeister, H., Mandel, L.** Extending UML to Model Navigation and Presentation in Web Applications. *In Modeling Web Applications, Workshop of theUML´2000. Ed. Geri Winters and Jason Winters, York, England*, October, 2000.

[16] **Landay, J. A., Borriello, G.** Design Patterns for Ubiquitous Computing. *Computer, Vol. 36(8)*, 2003, 93–95.

[17] **Link, S., Schuster, T., Hoyer, P., Abeck, S.** Focusing Graphical User Interfaces in Model-Driven Software Development. *In Proceedings of First International Conference on Advances in Computer-Human Interaction*, 2008, 3–8.

[18] **Martinez, A., Estrada, H., Sánchez, J.** From Early Requirements to User Interface Prototyping: A methodological approach. *17th IEEE International Conference on Automated Software Engineering 2002, Edinburgh, UK*, September 23–27, 2002.

[19] **Moreno, N., Fraternali, P., Vallecillo, A.** WebML modelling in UML. *Software, IET, Vol. 1(3),* June 2007, 67–80.

[20] OMG Unified Modeling Language (OMG UML), V2.2, OMG, 2009.

[21] **Palsberg J., Jay C.B.** The Essence of the Visitor Pattern. *Compsac, pp.9, 22nd International Computer Software and Application Conference*, 1998

[22] Petra, http://petra.cleverlance.com/.

[23] **Pinheiro da Silva, P., Paton N. W.** UMLi: the Unified Modeling Language for Interactive Applications. *In UML 2000 - The Unified Modeling Language. Advancing the standard. Third International Conference*, Springer, 2000, 117–132.

[24] **Pinheiro da Silva, P., Paton N. W.** User Interface Modeling with UML. *In Proceedings of the 10th European-Japanese Conference on Information Modelling and Knowledge Representation, Saariselk¨a, Finland, IOS Press*, May 2000.

[25] **Pinheiro da Silva, P., Paton, N. W.** A UML-Based Design Environment for Interactive Applications. *In Proceedings of UIDIS'01, Zurich, Switzerland, IEEE Computer Society*, May 2001, 60–71.

[26] **Preece, J., Rogers, H., Benyon, D., Holland, S., Carey, T**. Human-Computer Interaction. *Addison Wesley*, 1994.

[27] **Propp, S., Buchholz, G., Forbrig, P.** Task Model-Based Usability Evaluation for Smart Environments. *Engineering Interactive Systems*, LNCS, Vol. 5247, 2008, 29–40.

[28] Ribs: ReportMill's Interface Builder for Swing, http://www.reportmill.com/ribs/.

[29] Screen Architect, http://www.screenarchitect.com/.

[30] **Silingas, D., Butleris, R.** Towards Implementing a Framework for Modeling Software Requirements in MagicDraw UML. *Information Technology and Control, vol. 38(2)*, 2009, 153–164.

[31] **Silingas, D., Vitiutinas, R., Armonas, A., Nemuraite, L**. Domain-Specific Modeling Environment Based on UML Profiles. *In Information Technologies' 2009: Proceedings of the 15th International Conference on Information and Software Technologies, IT 2009, Kaunas, Lithuania, April 23-24, 2009, Kaunas University of Technology, Kaunas, Technologija*, 2009, 167–177.

[32] **Simon, J.** Interaction Happens: Prototyping Techniques. http://today.java.net/pub/a/today/2005/08/23/ prototyping.html, 2005.

[33] **Skersys, T.** Business knowledge-based generation of the system class model**.** *Information technology and control, Vol. 37(2)*, 2008, 145–153.

[34] **Tick, J**. Software User Interface Modeling with UML Support. *Institute of Software Engineering, IEEE*, 2005, 325–328.

[35] User Interface Markup Language (UIML) Version 4.0. Committee Draft. *OASIS*, 23 January 2008.

[36] Visual Paradigm: User Interface Designer, http://www.visual-paradigm.com/highlight/uidesigner2.jsp

[37] XUL: The XML User Interface Language, https://developer.mozilla.org/en/XUL.

[38] **Zhou, J., Stålhane, T.** A Framework for Early Robustness Assessment. *In Software Engineering and Applications (SEA' 04), MIT Cambridge, MA, USA*, 2004, 64–69.