Xplain: an Editor for building Self-Explanatory User Interfaces by Model-Driven Engineering

Alfonso García Frey, Gaëlle Calvary, Sophie Dupuy-Chessa

Xplain: an Editor for building Self-Explanatory User Interfaces by Model-Driven Engineering

Alfonso García Frey, Gaëlle Calvary and Sophie Dupuy-Chessa
University of Grenoble, CNRS, LIG
385, avenue de la Bibliothèque, 38400, Saint-Martin d'Hères, France
Alfonso.Garcia-Frey@imag.fr

ABSTRACT
Modern User Interfaces (UI) must deal with the increasing complexity of applications in terms of functionality as well as new properties as plasticity. The plasticity of a UI denotes its capacity of adaptation to the context of use while preserving its quality. The efforts in plasticity have focused on the (meta) modeling of the UI, but the quality remains uncovered. This paper describes an on-going research that studies a method to develop Self-Explanatory User Interfaces as well as an editor that implements this method. Self-explanation makes reference to the capacity of a UI to provide the end-user with information about its rationale (which is the purpose of the UI), its design rationale (why is the UI structured into this set of workspaces?, what's the purpose of this button?), its current state (why is the menu disabled?) as well as the evolution of the state (how can I enable this feature?). Explanations are provided by embedded models.

Author Keywords
Self-Explanatory User Interfaces, UI quality, help, design rationale, model-driven engineering, model transformation.

ACM Classification Keywords
H.5.2 User Interfaces: Theory and method.

General Terms
Design, Human Factors

INTRODUCTION

Problem Description

On the one hand, most software is too hard to use."Modern applications such as Microsft Word have many automatic features and hidden dependencies that are frequently helpful but can be mysterious to both novice and expert users" [16]. Users may require assistance while interacting with a User Interface (UI). Ideally, the UI must guide the user in accomplishing a task the application was designed for. The user can request help about functionality, features, or any information about the process of the task that is being performed. The UI must be able to provide the correct answer, giving the necessary information to the user in an appropriate format. This can take place at any time in the whole interaction process between both the user and the UI. However, modern applications cover only a few questions the user may have, or provide a general help instead of a clear and concise answer to a given question. Furthermore, help is created ad-hoc, this is, it has been previously generated and it's not able to cover new questions at run-time because they were not considered by the designers. UI design problems are not covered at all because the designers are not aware of them.

Moreover, the UI must deal with users having different levels of expertise. Even many long-time users never master common procedures [6] and in other cases, users must work hard to figure out each feature or screen [6].

The problem is greater for Plastic UIs [5, 22]. In Human-Computer Interaction (HCI), plasticity refers to the ability of UIs to withstand variations of contexts of use while preserving usability. The adaptation of the UI has been addressed using many different approaches over the years, including Machine Learning [8], Model-Driven Engineering (MDE) [18], and Component-oriented services [17]. Regardless of the approach, the tendency has been to focus on functional aspects of adaptation, while neglecting the usability dimension. Plastic UIs demand dynamic adaptation also for help systems because from now on, developers can't afford to consider all the different contexts of use one by one coding all possible ad-hoc solutions by hand. This variety of contexts of use complicates the prediction of the result and the final quality, making the design choices difficult.

As a result, dynamic solutions are required also for help systems. These help systems must now be aware of the context of use (user, platform and environment), the task, the structure and presentation of the UI.

MDE and MB-UIDE approaches

On the other hand, Model-Driven Engineering (MDE) exists since long time ago and is recently applied to the engineering of UIs. It consists in describing different features of UIs (e.g., task, domain, context of use) in models from which a final UI is produced [21] according to a forward engineering process. MDE of UI is assumed to be superior to the previous Model-Based User Interface Development Environment versions since it makes the UI design knowledge explicit, and external for instance as model-to-model transformations and

EICS'10, June 19–23, 2010, Berlin, Germany.

Figure 1. A help message leads the user through the UI.

model-to-code compilation rules [2]. However, whatever the apporach is (MDE or MB-UIDE), none of the automatically generated UI have enough quality, forcing designers to manually tweak the generated UI code [2]. Design knowledge can not be always explicitly represented into the models, but it has a potential to help users. Some of the models of the MDE approach as for instance the task model have this potential explicitly represented, and they can contribute also to guide and help the user.

Self-Explanatory User Interfaces

This paper focuses on an on-going research about Self-Explanatory User Interfaces (SE-UIs) by Model-Driven Engineering (MDE). Self-explanation makes reference to the capacity of a UI to provide the end-user with information about the rationale of the UI (which is the purpose of the UI), its design rationale (why is the UI structured into this set of workspaces?, what's the purpose of this button?), its current state (why is the menu disabled?) as well as the evolution of the state (how can I enable this feature?).

In Figure 1 a message is displayed by request when the user asks information about the window in the background. SE-UIs aim to generate these answers dynamically, taking into account the current context of use in which the interaction between the end-user and the UI is taking place.

To accomplish this research we study how SE-UIs can be built by MDE. The MDE approach allows us to separate the different concerns of the UI in several levels of abstraction. The Cameleon Reference Framework [4] illustrates this process. Some models of MDE as for instance the task model have part of the design knowledge explicitly represented, and they can contribute also to guide and help the user.

In this paper we present two different aspects of our research. First, we explain a method to build SE-UIs using the MDE approach. In a second time, we propose an editor that implements this method in a visual way. The editor presented in this work is a prototype of the future editor we aim to build. Even if some of the features are not available at the time of publishing this work, the general idea of the method and details about its implementation are provided in this paper.

RELATED WORK

Two major areas are involved in our Self-Explanation approach: MDE and UI quality. The next two sections set up the bases of our contribution regarding some of the related works in these fields.

MDE

The Cameleon Reference Framework [4] presented a MDE-compliant approach for developing UIs consisting of four different levels of abstraction: Task Model, Abstract User Interface, Concrete User Interface and Final User Interface. These levels correspond, in terms of MDE, to Computing-Independent Model (CIM), Platform-Independent Model (PIM), Platform-Specific Model (PSM) and the code level respectively. In the Model-Driven Development (MDD) many transformation engines for UI development have been created. Several researches have addressed the mapping problem for supporting MDD of UIs: Teresa [15], ATL [11], oAW [11] and UsiXML [20] among others. A comparative analysis can be found in [10]. Semantic Networks have been also covered for UIs [9]. The Meta-UI concept was firstly proposed in [7] and deeply explored later in many other works. In one of them [19], the concept of Mega-UI is studied introducing Extra-UIs, allowing a new degree of control by the use of views over the (meta-)models. We will focus on it later as these views are relevant for the explanation of the UI and consequently for the end-user's comprehension.

UIs Quality

Help systems have been extensively studied. One of the most relevant works is the Crystal application framework [16]. Inspired by the Whyline research [12], "Crystal" provides an architecture and interaction techniques that allow programmers to create applications that let the user ask a wide variety of questions about why things did and did not happen, and how to use the related features of the application without using natural language [16]. Even if this approach does not cover the capacity of adaptation to different contexts of use, it represents an important improvement in quality for the end-user in terms of achieved value. Quality can be improved regarding not only the *achieved value*, but also from the perspectives of *software features* and *interaction experiences* [13]. The integration of Usability Evaluation Methods (UEM) [14] into a MDA process has been proved to be feasible in [1]. In particular, the evaluation at the PIM or PSM should be done in an interactive way until these models have the required level of usability. Different UEMs (e.g., heuristic evaluation, usability test, etc.) can be applied iteratively until the concerned models have the required level of usability. A set of ergonomic criteria for the evaluation of Human-Computer Interaction (HCI) can be found in [3].

Our research improves quality of help systems allowing a new range of questions. Adaptation to the context of use is now considered since SE-UIs understand their own rationale.

THE METHOD

Basis

One of the ways to explore SE-UI involves the task model and its rationale. A task model describes the user's task in terms of objectives and procedures. Procedures recursively decompose tasks into subtasks until one or more elementary tasks are reached, i.e., tasks which would be decomposable into physical actions only ("press the button"). A task model is well-defined then by the following terms:

Nodes Containing abstract tasks

Leaves Special nodes containing elementary tasks

Branches Expressing logical and temporal relations between tasks, subtasks and elementary tasks

The explicit information contained into the branches can help and guide the end-user answering questions related to different aspects of the UI. For instance, regarding the rationale of the UI questions like *which is the purpose of the UI?* can be successfully answered; also, questions as *why is the UI structured into this set of workspaces?* or *what is the purpose of this button?* can be explained understanding the relations of the design rationale. The current state of the UI and consequently the state of the application, can trigger a different kind of questions to the end-user as for instance *why is the menu disabled?*, as well as questions related to the overall progress of a task or questions about the evolution of the current state of the application as for example *how can I enable this feature?* Answers for all of them can be obtained exploring tasks and subtasks (nodes), elementary tasks (leaves) and relations between them (branches) in the task model.

This research will consider also how different views of the model, in form of extra-UIs, can help the end-user to understand the UI. An extra-UI [19] is a UI which represents and gives the control of a UI through a model. These views can improve the end-user's comprehension as they are relevant for the explanation of the UI. Extra-UIs provide a new degree of control over the (meta-)models of the UI; both designer and end-user can see and understand how tasks are decomposed and represented in a specific UI. In other words, how the UI is interfacing the interaction between the application and the user. Designers can express this interaction in the form of relations between tasks and elements of the final UI with the method explained in the next section.

Four steps method

This work will explore a method to provide designers with a technique to add Self-Explanation to UIs specifying how end-user's tasks are directly related to the final UI level. The method consists in four steps:

1. Specify the final UI of the model-compliant application that it will be extended with SE functionality.
2. Define the task model of the application.
3. Specify the relations between both the task model and the final UI.

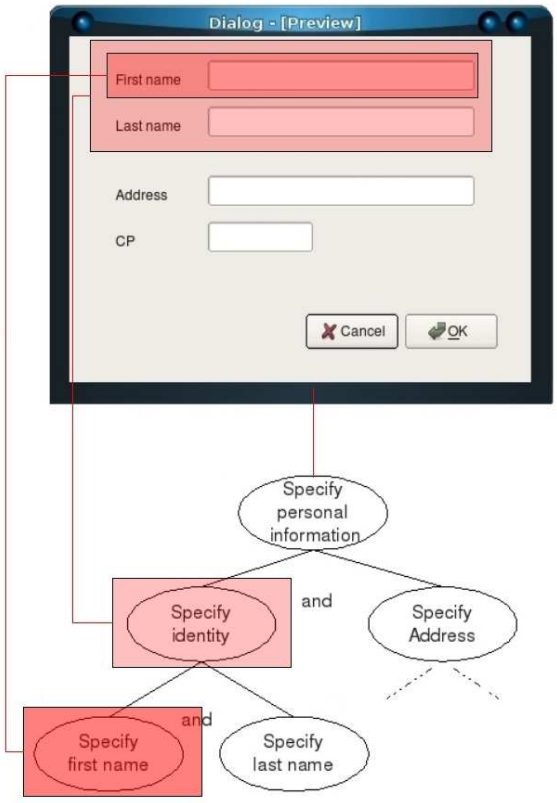

Figure 2. Association between UI and a task model.

4. A new final SE-UI will be generated from these relations, adding SE functionality in real-time.

The definition of the UI in the first step can be done considering several solutions. For this work, we use a specific file format definition in which a UI is expressed in a XML syntax-like. Other approaches and modalities can be considered as well.

For the second step we propose the use of the Concurrent Task Trees (CTT) notation to describe task models, which can specify a wide range of temporal relationships among tasks. CTT is a compact and graphical notation, immediate both to use and understand. Its logical structure makes it suitable for designing large sized applications even if, for the moment, we used a simplified version of the Concurrent Task Trees, which does not use the range of possible temporal operators.

Once we have specified the UI -by providing the specific file in our case-, and we have defined the task model of the associated application in the second step, we will link both of them in a third phase. Figure 2 shows an example of links between the given UI and the associated task model. Here, the task called *Specify identity* is visually connected to a group of widgets, containing two labels and two input fields. Then, the elementary task *Specify first name* which is also a sub-

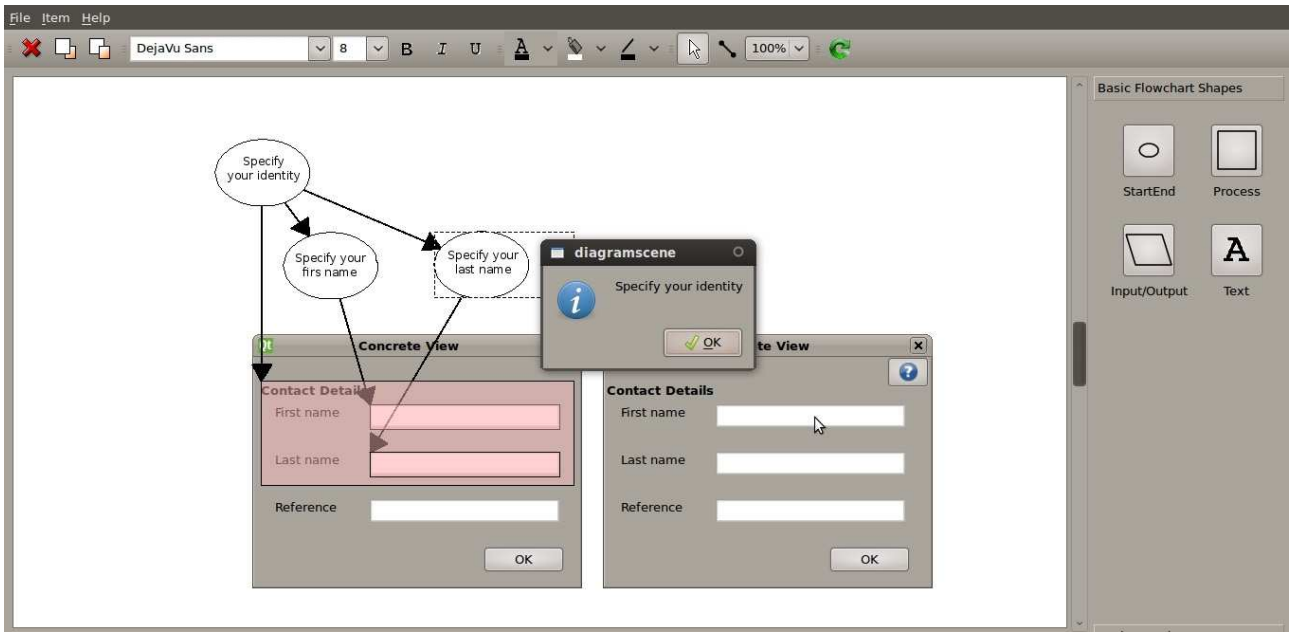

Figure 3. A Self-Explanatory UI is generated on the right side from the links on the left side.

task, is connected to a new subgroup of two widgets, one label and one input field.

Once all the links are established we can generate a UI improved with SE-UI.

The main advantage of linking UIs with task models is that from now on, designers have no need of a deeply comprehension of all the model-to-model and model-to-code transformations between all the four levels of MDE approach. A visual representation gives direct information about these relations because connections are explicitly represented in a visual render.

THE EDITOR

Xplain provides the designers with the ability to create SE-UI from a given a UI, by describing the task model of the associated application in a first step, then linking the states of this model to the desired components of the given UI, and finally generating a new UI based on the first one, but extended with SE functionality.

It is a graphical UI builder that allows designers to interactively specify all graphical aspects of an application, including both the widgets that go around the application windows, and the application-specific SE features as well. The designer can also specify the behaviors that these graphics exhibit creating relations between the widgets and the task graph model. From this information, Xplain generates help for the end-users.

As a first approach, help can be given with a help button. We can see an example inside the UI on the right side in figure 3. Other approaches can be considered as well. By clicking this help button, the application enters in a help mode where the end-user can ask about different elements of the UI just by clicking on them. Answers will be generated in real-time in different ways. The following section illustrates an example of this procedure.

Answering questions

This work will study also how different questions can be answered. The first approach will associate a description to each element (tasks, relations, widgets, etc.) of figure 2. Other approaches like semantic networks [9] can be considered in the future. If the end-user asks himself, for instance, *Why is the OK button disabled?*, by clicking on this button using the special help mode, the system can say that the task is not completed. In figure 1 the message is dynamically derived from the relations of figure 2. For an edit box, the application can say *You must fill in + Description of the task*, where *your personal information* is the description. A more specific information can be generated exploring the task model. For instance, we can check all the subtasks of the uncompleted task. In the example before, we can answer also that the user needs to fill in the first name and the last name, because these subtasks are both uncompleted.

Example of Use

In Figure 3 we can see the prototype in use. The designer has opened a .ui file that contains the specification of the UI in a XML-like syntax. This .ui format comes from the Qt development framework. Other formats should be also supported in the future. Once the designer opens the file, Xplain parses the contained data and the UI is rendered inside the workspace. In the right side of the editor, the designer has a panel showing different tools to create task models. At the moment of this publication the CTT notation is not supported and the designer can just express nodes and links between these nodes as we can see in Figure 3. To create the graph,

the designer can select the circle in the panel on the right and draw as many nodes as necessary into the workspace. Each time the designer creates a node, a pop-up message will ask him a description. The designer can then enter an associated description for this node and continue building new nodes. This description will be used later as an explanation for the associated widget. In other words, the description of a node is the description of the task represented by this node.

Once the task model is defined, the designer can pick up the linker tool in the toolbar on the top of the window and link with the mouse all the nodes with all the widgets of the UI. Different widgets can be grouped also under the same task. The designer can link more than one widget with the same node, grouping content by tasks.

Finally, the designer can press the green arrow at the right of the toolbar to generate a completely new UI enhanced with Self Explanation. As explained before, a little help button is added to the new window in order to let the user ask about the UI.

This new UI is running into the workspace and the designer can enter data, use the new button to test the new functionality, and test other properties of the UI without the need of an external tool to recompile the generated ui file.

The prototype has been coded in C++ using the Qt framework.

WORK IN PROGRESS AND FUTURE WORK

This prototype brings out some questions for SE-UIs:

1. Can complementary views of the UI help us to guide the end-user?
2. How and in which manner?
3. Which part or parts of the MDE approach need to be visible to the end-user?
4. Which is the best way to collect information from the end-user about the encountered problems while performing a task?

Questions one to three can be addressed also to the designers. Due to these questions we continue to develop the state of the art. For example, we are also exploring different ways to let the user ask about the UI and how to deal with the possible answers.

Other further work is to take into account all the UsiXML models. This will help us not only with plasticity but also with modality, allowing us to create more sophisticated SE-UIs.

The prototype does not support models at the moment and the designer must create the associated graphs by hand. The integration of the models as well as the CTT notation for the task model is a priority for the near future.

CONCLUSION

This research aims at improving the quality of UIs. It explores MDE of UIs to provide Self-Explanation at run-time, taking benefit from the four levels of abstraction and their relationships.

A prototype of the SE-UI editor is being developed to support designers. This prototype implements a four steps method that takes a UI in input and generates the associated SE-UI in output. The designer can interact with the generated SE-UI directly inside Xplain. This prototype lets different questions open that motivate our research in different ways.

Finally complementary views of the UI needs to be explored in order to exploit the models of the MDE approach, explaining the UI itself and giving to the user a new dimension of control by these views. This opens the work on End-User programming.

ACKNOWLEDGMENTS

This work is funded by the european ITEA UsiXML project.

REFERENCES

1. S. Abraho, E. Iborra, and J. Vanderdonckt. *Maturing Usability*, chapter Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool, pages 3–32. Human-Computer Interaction Series. Springer-Verlag, 2008.
2. N. Aquino. Adding flexibility in the model-driven engineering of user interfaces. In *EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, pages 329–332, New York, NY, USA, 2009. ACM.
3. J. C. Bastien and D. L. Scapin. Ergonomic criteria for the evaluation of human-computer interfaces. 0 RT-0156, INRIA, 06 1993.
4. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting With Computers Vol. 15/3*, pages 289–308, 2003.
5. B. Collignon, J. Vanderdonckt, and G. Calvary. Model-driven engineering of multi-target plastic user interfaces. In *Proc. of 4th International Conference on Autonomic and Autonomous Systems ICAS 2008*, pages 7–14, 2008. D. Greenwood, M. Grottke, H. Lutfiyya, M. Popescu (eds.), IEEE Computer Society Press, Los Alamitos, Gosier, 16-21 March 2008.
6. M. Corporation. Microsoft inductive user interface guidelines, 2001. http://msdn.microsoft.com/en-us/library/ms997506.aspx.
7. J. Coutaz. Meta-user interfaces for ambient spaces. In *Tamodia'06*, 2006. 8 pages.
8. J. Coutaz, L. Balme, G. Calvary, A. Demeure, and J.-S. Sottet. An MDE-SOA approach to support plastic user interafces in ambient spaces. In *Proc. HCI International 2007*, pages 152–171, 2007. Beijing, July 2007.

9. A. Demeure, G. Calvary, J. Coutaz, and J. Vanderdonckt. Towards run time plasticity control based on a semantic network. In *Fifth International Workshop on Task Models and Diagrams for UI design (TAMODIA'06)*, pages 324–338, 2006. Hasselt, Belgium, October 23-24, 2006.

10. J. González Calleros, A. Stanciulescu, J. Vanderdonckt, D. J.P., and M. Winckler. A comparative analysis of tranformation engines for user interface development. In *Proc. of the 4th International Workshop on Model-Driven Web Engineering (MDWE 2008)*, pages 16–30, Tolouse, France, 2008. CEUR Workshop Proceedings.

11. F. Jouault and I. Kurtev. Transforming models with atl. In *Satellite Events at the MoDELS 2005 Conference*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138, Berlin, 2006. Springer Verlag.

12. A. J. Ko and B. A. Myers. Designing the whyline: a debugging interface for asking questions about program behavior. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 151–158, New York, NY, USA, 2004. ACM.

13. E. Lai-Chong Law, E. T. Hvannberg, and G. Cockton. *Maturing Usability. Quality in Software, Interaction and Value*. Human-Computer Interaction Series. Springer-Verlag, 2008.

14. E. L. Law, E. T. Hvannberg, G. Cockton, P. Palanque, D. Scapin, M. Springett, C. Stary, and J. Vanderdonckt. Towards the maturation of IT usability evaluation (MAUSE). In *Human-Computer Interaction - INTERACT 2005*, pages 1134–1137. 2005.

15. G. Mori, F. Paterno, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, 30(8):507–520, 2004.

16. B. A. Myers, D. A. Weitzman, A. J. Ko, and D. H. Chau. Answering why and why not questions in user interfaces. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 397–406, New York, NY, USA, 2006. ACM.

17. P. D. S. Paulo. User interface declarative models and development environments : A survey. 2007.

18. S. N. G. F. Sheshagiri, M. Using semantic web services for context-aware mobile applications , in proceedings of acm mobisys2004 workshop on context awareness, boston, massachusetts, usa, june 2004. 2004.

19. J.-S. Sottet, G. Calvary, J.-M. Favre, and J. Coutaz. *Megamodeling and Metamodel-Driven Engineering for Plastic User Interfaces: Mega-UI*. 2007.

20. J. Vanderdonckt. A MDA-Compliant environment for developing user interfaces of information systems. In *Advanced Information Systems Engineering*, pages 16–31. 2005.

21. J. Vanderdonckt. Model-driven engineering of user interfaces: Promises, successes, failures, and challenges. In *Proc. of 5th Annual Romanian Conf. on Human–Computer Interaction ROCHI'2008, (Iasi, 18–19 September 2008), pp. 1—10. Matrix ROM, Bucarest*, 2008.

22. J. Vanderdonckt, J. Coutaz, G. Calvary, and A. Stanciulescu. *Multimodality for Plastic User Interfaces: Models, Methods, and Principles*, chapter 4, pages 61–84. 2008. D. Tzovaras (ed.), Lecture Notes in Electrical Engineering, Springer-Verlag, Berlin, 2007.