A Framework for Incorporating Usability into Model Transformations

Xulin Zhao
Queen's University
Department of Electrical and Computer Engineering
Queen's University
Kingston, Ontario, Canada, K7L 3N6
4xz5@queensu.ca

Ying Zou
Queen's University
Department of Electrical and Computer Engineering
Queen's University
Kingston, Ontario, Canada, K7L 3N6
ying.zou@queensu.ca

ABSTRACT
The usability of user interfaces is crucial for the success of an application. Model driven user interface (UI) development speeds up the production of UIs and improves the maintainability of UIs. However, the usability evaluation of UIs is usually conducted by end-users or experts after UIs are generated. Such a user centric evaluation is usually time consuming and expensive, especially when the usability problems are detected in the last phase of the software development. In this paper, we propose a framework that incorporates the usability evaluation as an integral part of automatic processes for UI generation. To link the usability goal into the UI generation process, we model the usability using a goal graph for each intermediate UI model and associate the usability goals to the attributes of the models. Our proposed framework detects and addresses usability problems in the early phase of the software development.

1. INTRODUCTION
The user interface (UI) of an application provides an interactive environment for a user to communicate with the system. The usability of UIs directly affects users' satisfaction of an application. The success of an application depends on the usability of its UIs [1]. Surveys [2,3] show that the UI with poor usability is the major reason of frustration. However, the UI development is time consuming and expensive. On average, 50% of the development time is devoted to developing the UIs of an application [4]. Approaches, such as UI management system (UIMS) [5] and model based UI system [5], are proposed to improve the development of UIs. A great deal of tools is developed to assist the efficient development of UIs [5]. However, studies [4, 8, 9] indicate that the evaluation of usability is not well addressed due to the pressure of product deliverables and limited budget. Usability evaluation techniques such as heuristic evaluation, cognitive walkthrough, user testing and metrics (e.g., success rate, and error rate [10]) can only be conducted after the UIs are fully developed. Even though the GOMS (goal, operator, method and select rules) [11] approach does not require UI prototypes, it can only perform the evaluation on the detailed task specification which is often manually created.

In our work, we aim to improve the usability of UIs using model driven software development techniques. In our previous work [12], we apply model driven development techniques to automatically generate UIs from business process definitions. Specifically, we use three declarative models, including task models, dialog models and presentation models to incrementally generate UIs. The task models are used to model the functionalities performed by users, in order to accomplish the business objectives specified in the business process definitions. The dialog models define users' interaction with the system. The presentation models describe the visual appearance of UIs. We define rules to accomplish the transformation between models of different abstraction levels and generate UIs incrementally. Moreover, we apply usability models to guide the transformations. We interpret users' usability goals to model constraints and use these constraints to guide the selection of transformation rules. Eventually, we generate UIs that conform to users' usability expectations.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 provides an overview of the proposed framework. Section 4 illustrates the proposed framework using an example. Section 5 concludes the paper and presents our plans for the future.

2. RELATED WORK
A lot of researchers have addressed the problem of optimal UI generation. Gajos *et al.* presented an approach of rendering user interfaces in [6]. In this approach, the UI render requires three inputs, including a UI specification, a device model, and a user model. A UI specification is composed of a set of UI elements and a set of UI constraints. The device model contains available widgets, constraints, and suitability evaluation functions of a specific display device. The user model is defined by a set of user events. A user event is comprised of a user element and their value variations in response to users' interactions. The rendering algorithm generates UIs with the minimal user effort. This approach treats UI generation as an optimization problem and can easily generate UIs for different devices. The UI specifications used in this approach are equivalent to abstract presentation models. In our research, we incorporate usability into model based UI design and automatically generate UI models and prototypes from business processes. In our work, the proposed framework can generate different UI alternatives according to different user models and can be used as a preparation for the approach in [6].

Sottet *et al.* have addressed the problem of preserving usability during UI adaption [7]. This paper stands for the use of adviser tools and classifies model transformations into two types: predictive models and descriptive models. Predictive models can reform UI models without ambiguous (e.g. $f(x)=x+2$ [7]). Descriptive models are qualifiers (e.g. $f(x)>x$ [7]). The reservation of usability is achieved by well designed transformation models. In our approach, we define our transformation models using transformation rules. All transformation rules are predictive. In addition, we leverage usability goal graphs to model users'

usability expectations. Different users and devices generally have different usability requirements. Model constraints are mostly descriptive and can be changed during the UI development process. In the early stage of UI development, we generally are not clear about users' expectations. In this case, we can set the boundaries of model constraints using empirical values. For example, we can limit the number of widgets in a window to the range from 3 to 20. As the development process advance, we will make clear about users' requirements. Then, we can set specific values to model constraints.

3. INCORPORATING USABILITY GOALS INTO MODEL TRANSFORMATIONS

The goal of the proposed framework is to decrease the cost of usability assessment of UI generation. The framework is depicted in Figure 1. Functional requirements and usability goals are modeled as constraints for the generated models. A set of transformation rules is defined to generate models with different levels of details and incrementally transform these models towards the final code. We can select transformation rules that conform to the constraints on models and generate the final UI with high usability. The rule selection and model generation are iterative process and continue till the usability goals are achieved. In the following subsections, we illustrate the proposed framework that uses usability goals to guide the selection of transformation rules.

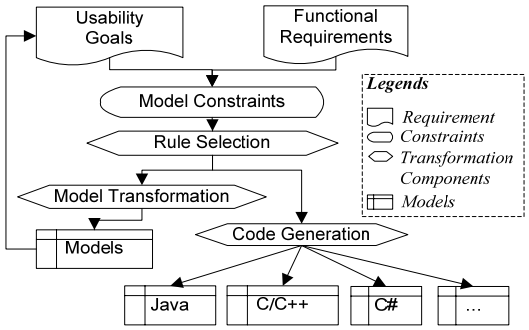

Figure 1: A Framework for Incorporating Usability Goals into Model Transformations

3.1 Associating usability goals to the constraints of the models

In our research, we use model constraints (e.g., the number of widgets in a window is less than 20) to bridge the gap between the usability goals and the transformation rules. We represent the associations between quality goals and model constraints using goal graphs and hierarchical quality models [14]. Generally, a goal is an objective or a desired state that a system can achieve. In the domain of software engineering, the goals are used to elaborate functional and non-functional requirements [8,13]. A goal is analyzed and decomposed into a set of sub-goals, which have the AND- or OR-dependencies with the parent goal. We can produce a dependency graph to represent the dependencies between the goal and its sub-goals. As depicted in Figure 2, an example usability goal graph is used to guide the generation of the presentation models. As specified in ISO/IEC 9126 [14], a usability can be achieved by a set of sub-goals, such as memorability, learnability, and low error rate [10]. Each sub-goal can be further broken down to lower level goals. For example, learnability is affected by sub-goals, such as the complexity [15], consistency [17], and the structure of information grouping [17]. For the usability goals, the leaves are a set of metrics used to directly or indirectly measure the usability or its sub-goals [8]. For example as shown in Figure 2, the complexity of a UI can be directly measured by a set of metrics, such as the number of widgets in a window. The result of the complexity metrics indirectly indicates the usability of the UI.

3.2 Rule selection

Rules are designed to transform one or more properties of the source models to a target model. From the usability point of view, applying a transformation rule must have certain impact on the usability of the target models. For example, verifying a shopping cart and checking out are two UI distinct operations in an on-line shopping application. The two operations use the same data as inputs (i.e., shopping cart information). These operations can be implemented into two separate windows which have their own copy of the shopping cart information. In this case, the user has to review the shopping car information twice when performing these two operations. Alternatively, a transformation rule can be defined to merge these two windows into one window if both windows display the same information (e.g., shopping cart) required for triggering two distinct operations (e.g., verify the shopping cart and checkout). This merging transformation rule reduces the number of windows in a UI, and consequently improves the usability by removing the redundant interactions between the user and the application. To decide the most proper transformation rules to apply, we evaluate the impact on the usability before applying a rule to the source model. More specifically, we consider the following two heuristics for selecting a transformation rule:

1) If the usability attributes of a model cannot meet the specified usability goals, we select the transformation rules that can improve the usability of the model. For example, it is more efficient for a user to complete a business process through fewer window transitions. Therefore, we must apply transformation rules that can produce the shortest path length for fulfilling a task.

2) The selected transformations have no conflicts with previously applied transformations. The usability attributes are defined within a desirable range. For example, the number of widgets in a window generally needs to be less than 20 widgets [18]. If the application of a transformation makes a usability attribute fall below the desired range, we say that the transformation rule has conflicts with the previously applied transformation rules. If a rule has no negative impact on a usability attribute or if the negative impact is within the acceptable range, we consider that the transformation rule has no conflicts with the previously applied transformations. For example, adding widgets (e.g., buttons or text fields) to a window will increase the number of widgets in a window. If the total number of widgets is still less than 20, this transformation is acceptable.

4. AN EXAMPLE

In this section, we use an example to demonstrate the steps that generate the UI code from the presentation model in the proposed framework. We develop a prototype tool that guides the UI generation using the usability goal graph and evaluates the

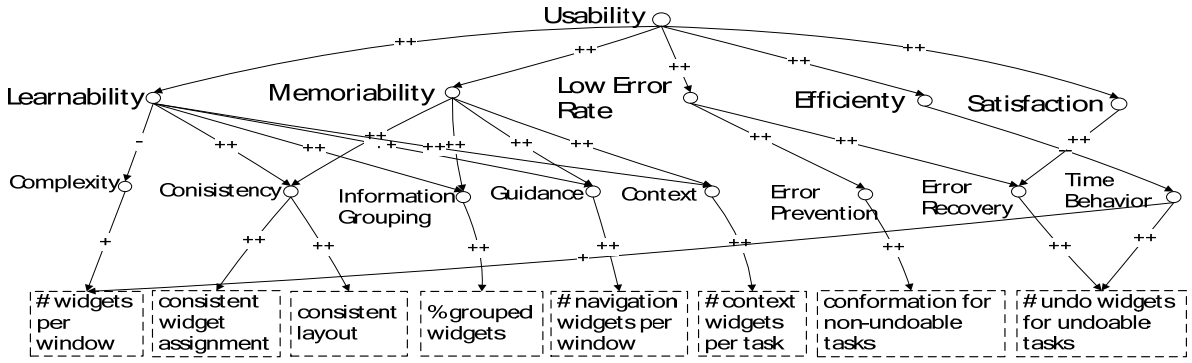

Figure 2: An Example Usability Goal Graph

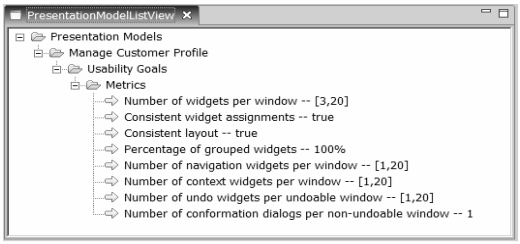

Figure 3: Translated constraints

usability of generated models (e.g., code). Each model has constraints with the acceptable thresholds for the metrics (i.e., the leaves in the usability goal graph). For example, we set the upper bound of the number of widgets per window to 20. We measure the usability goals using a set of metrics, as shown in Figure 3.

To generate source code of the UI, we apply transformations consecutively to move the initial business process definitions towards the code. As a first step of generating code from the presentation model, we want to ensure that the structure of the generated UI can ease a user to navigate thought the UI. For example, a window of high usability should have a navigational widget and a contextual guidance widget to support a user to fulfill business tasks. A navigational widget guides a user to transit between the windows for fulfilling the business tasks. A contextual guidance widget displays the relevant information to assist a user in performing the current task. For example, a user may purchase a book online. It is convenient for the user to search for the book if we can display all the relevant books for the user in a window (i.e., widget). In addition to the widgets that are used for the user to complete the required tasks. The additional widgets can be added to guide the navigation and the task fulfillment in order to improve the user experience. In this case, the lower bound of the number of widgets for each window can be set to three widgets because a window should have three widgets for the improved usability, including one widget that allow a user performs the task, one navigational widget, and one contextual guidance widget. By considering the usability goals (e.g., providing more guidance to the user), an appropriate transformation rule is selected to define the navigational structure of the UI and the major components in each window.

To improve the usability of each window, we use the well-designed task patterns to implement the business tasks performed in each window. The task patterns document the major components needed for implementing a task (e.g., searching a product, and creating a shopping cart). For example,

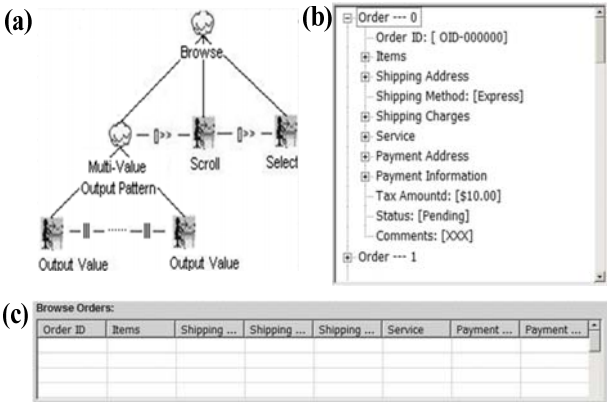

Figure 4: (a). The structure of a browse task pattern,
(b). the tree implementation of the browse pattern,
(c). the table implementation of the browse pattern.

Figure 4 (a) is a browse task pattern [16] in the Concurrent Task Tree [19] format. This pattern describes the major components to implement a browsing functionality. The structure of the browser task pattern can be broken down into more detailed components, including a multiple value form that displays one or more results, a scroll bar that allows a user to scroll down the result list and view the results, and a selection widget that enables a user to select the result of interest.

Furthermore, we need to determine appropriate widgets to implement the components defined in the browse pattern. Multiple alternative widgets can be used to implement the multi-value output pattern in the browse pattern. Different widgets have different impact on the complexity of the generated windows. In the simplest form, each output value can be displayed using one label widget. The number of the output values is dependent on a particular context of the application. As the number of the output values increases, the more label widgets are needed to display the output values. Consequently, the complexity of the UI increases as the number of widgets grows. To keep the UI simple, a transformation rule can be defined to transform the multi-value patterns into a structured widget with defined boundaries. Figure 4 (b) and Figure 4 (c) are two possible implementations of the browse task pattern. Figure 4 (b) implements the pattern using a tree widget and Figure 4 (c) implements the browse pattern using a table widget.

To ensure that the generated windows follow the consistent look-and-feel for the same task the applications, the transformations select the most appropriate widgets that implement a task pattern and apply this selection throughout the UI generation process. By comparing Figure 4 (b) with Figure 4 (c), we can observe that a table widget is more suitable to implement the browse pattern than a tree widget, for the reason that a user can read all output values at a glance in a table widget. Therefore, all browse patterns are transformed to table widgets patterns as a default transformation rule. A UI developer can customize the default widgets of task patterns.

The generated UI may not be able to achieve all of the usability sub-goals. We generate alternative UIs based on different selection of transformation rules. The prototype tool automatically computes the usability metrics to estimate the achievement of the goal for each UI alternative. For example as shown in Figure 5, the UI for each business process can be analyzed to estimate the level of conformance to the usability goal.

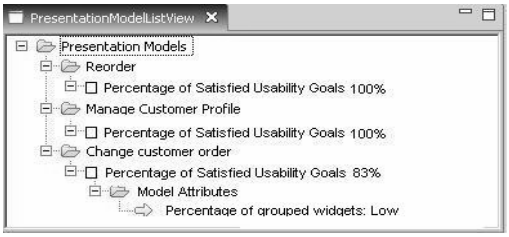

Figure 5: Sample Results for Rule Selection

5. CONCLUSION AND FUTURE WORK

In this paper, we present a framework for incorporating the usability goals into the model transformations. We aim to evaluate the usability in the early stage of model transformation instead of deferring the usability evaluation when the UI is fully developed. We develop a prototype to demonstrate the feasibility of the proposed framework. An example is used to illustrate the result of our prototype tool. We use a set of metrics to evaluate the usability of the generated UI.

In the future, we plan to conduct more case studies to refine the proposed framework. We also want to recruit the users to evaluate the usability of the generated UI. The feedback from the users can help to refine the usability goal graphs and the selection of transformation rules. We will improve the customizability of the framework, so that users can integrate their own UI widgets, UI patterns, usability criteria and metrics into the UI generation process.

6. REFERENCES

[1]. D. Stone, C. Jarrett, M. Woodroffe and S. Minocha. User Interface Design and Evaluation, *Morgan Kaufmann, Los Altos, CA*, (2005) ISBN 0-12-088436-4.

[2]. P. Jackson, J. Favier and I. Stagia. Segmenting Europe's Mobile Consumers, *Forrester Research Technographics® Report* (2002).

[3]. M. Ramsay and J. Nielsen. WAP Usability Report, *Nielsen Norman Group* (2000).

[4]. BA. MYERS, and MB. ROSSON. Survey on user interface programming. *In Human Factors in Computing Systems, In Proc. of SIGCHI ACM, New York (1992)*, pp. 195–202.

[5]. B. Myers SE. Hudson, and R. Pausch. Past, Present, and Future of User Interface Software Tools, *ACM TOCHI* (2000), no. 1, pp. 3-28.

[6]. K. Gajos and DS. Weld. SUPPLE: Automatically Generating User Interfaces. *In Proc. of IUI* (2004), pp. 93.

[7]. JS. Sottet, G. Calvary, and JM. Favre. Mapping Model: A First Step to Ensure Usability for sustaining User Interface Plasticity. *2nd Int. Workshop on MDDAUI, CEUR Proceedings* (2006), vol. 214

[8]. Y. Zou and K. Kontogiannis. Migration to Object Oriented Platforms: A State Transformation Approach, *In Proc. of the 19th IEEE Int. Conf. on Software Maintenance* (2002)

[9]. L. Bass, BE. John, N. Juristo, and MI. Sanchez. Usability-supporting Architectural Patterns, *In Proc. of 26th Int. Conf. on Software Engineering ICSE* (2004), pp. 716-717.

[10]. J. Nielsen. Usability Engineering, *Academic Press* (1993).

[11]. BE. John and DE. Kieras. The GOMS family of analysis techniques: Comparison and contrast, *ACM TOCHI* (1996), pp. 320–351

[12]. X. Zhao, Y. Zou, J. Hawkins and B. Madapusi. A Business Process Driven Approach for Generating E-Commerce User Interfaces, *ACM/IEEE 10th Int. Conf. on Model Driven Engineering Languages and Systems* (2007).

[13]. P. Giorgini, J. Mylopoulos, E. Nicchiarelli and R. Sebastiani1. Reasoning with Goal Models, *In Proc. of the 21st Int. Conf. on Conceptual Modeling* (2002).

[14]. ISO/IEC9126: Information Technology - Software Product Quality. Part 1: Quality Model (1998).

[15]. MF. Bertoa, JM. Troya and A. Vallecillo. Measuring the Usability of Software Components, *Journal of Systems and Software* (2006), vol. 79, no. 3, pp. 427-439.

[16]. D.Sinnig, H. Javahery, P. Forbrig,, and A. Seffah. The Complicity of Model-Based Approaches and Patterns for UI Engineering, *In Proc. of BIR'03* (2003), pp. 120-131

[17]. S. Abrahão and E. Insfran. Early Usability Evaluation in Model Driven Architecture Environments, *Sixth Int. Conf. on Quality Software* (2006), pp. 287-294

[18]. M. Elkoutbi and RK. Keller. User Interface Prototyping Based on UML Scenarios and High-Level Petri Nets. *In Proc. of ICATPN* (2000): LNCS 1825, Pages 166–186

[19]. F. Paternò. Model-Based Design and Evaluation of Interactive Applications. *Springer Verlag*, ISBN 1-85233-155-0 (2000)