A Business-Process-Driven Approach for Generating E-Commerce User Interfaces

Xulin Zhao, Ying Zou
Electrical and Computer Engineering Depart.
Queen's University
Kingston, Ontario, Canada
4xz5@qlink.queensu.ca, ying.zou@queensu .ca

Jen Hawkins, Bhadri Madapusi
IBM Toronto Laboratory
IBM Canada Ltd.
Markham, Ontario, Canada
{jlhawkin, bhadrim} @ca.ibm.com

Abstract. A business process contains a set of interdependent activities that describe operations provided by an organization. E-commerce applications are designed to automate business processes. A business process specification (i.e., a workflow) is defined by a business analyst from the viewpoint of the end-users. The process encapsulates the knowledge related to the natural work rhythms that a business user would follow when using an e-commerce application. In this paper, we analyze the information embedded in business process specifications, and infer the functional and usability requirements. We use the inferred information in a model-driven approach to automatically generate user interfaces (UIs) from a business process specification through a set of transformations. To improve the usability of UIs for the e-commerce applications, each transformation is guided by usability principles.

Keywords: Business process, User interface generation, Usability, Model driven engineering, Task model, Dialog model, Presentation model

1 Introduction

A business process is a set of linked activities that need to be carried out to achieve business goals for an organization. These activities are called tasks and may be performed by multiple roles. For example, *"Find product"* and *"Validate credit card"* are two tasks in an online shopping business process and are performed by the role "*Customer sales representative*" (CSR) and the role "*Bank clerk*", respectively. E-commerce applications are designed to automate business processes in an organization [1]. In today's fast-changing business environments, marketing strategies and technical innovations drive the evolution of business processes. To maintain their competitive edge, organizations must keep on updating the business logic and UIs of e-commerce applications in order to reflect new business initiatives. However, the updates and evolution of UIs are often performed without referencing the underlying business processes. Therefore, the UIs of e-commerce applications and the business processes rarely evolve consistently.

The usability of UIs of e-commerce applications is crucial for the survival of a business and its success. The key to attract customers is e-commerce applications with high quality content, ease of use, quick response and frequent updates. However, studies [2] show that the majority of the UIs of e-commerce applications suffer from

usability problems. For example, applications often provide more functionalities than necessary. It is not obvious for users to navigate through a data centered UI in order to fulfill business tasks. Approaches have been proposed for automating the design and implementation of UIs [3, 5]. However, these approaches create the designs of the UIs from the perspectives of the software development domain. Without the participation of business users, the UI might not satisfy the usage patterns of business users. Therefore, it is crucial to provide an approach that can directly involve the business user in the design of UIs in order to ensure the usability of these interfaces.

In this work, we propose a framework that can automatically generate UIs from business processes using a sequence of model transformations. More specifically, we analyze business processes to obtain the functional requirements specified by the business analysts. An initial task model is generated to reflect the functionalities in the UI. We extract contextual information from business process specifications to provide supports for users. Examples of contextual information are the order of task execution, the data flow from one task to another, and the pre-conditions and post-conditions for task execution. To generate UIs with different quality goals (e.g., high learnability and high efficiency), we apply usability practices and UI design principles to guide the transformations of business processes into UI design models. For example, UIs with high learnability can provide more guidance for the users. Developers can pick their preferred UI based on their quality requirements.

In the following sections, we introduce the details of our approach. Section 2 describes other researchers' work related to our approach. Section 3 introduces our model-driven UI generation framework. Section 4 presents models and their transformations. Section 5 illustrates our approach using case studies. Section 6 presents our conclusion and future work.

2 Related Work

To improve the efficiency in software development, approaches for automatic UI generation were proposed to separate UIs from business logic [5]. Model based UI development typically creates a series of declarative models, such as models of user tasks, dialogs and presentations [6]. User task models are used to elicit the requirements of functionalities. Derived from the identified user tasks, dialog models describe the interactions between the user and the UI. Furthermore, presentation models are created by assigning abstracted graphic user interface (GUI) widgets to dialog models. Finally, UIs are implemented by mapping presentation models to specific UI implementation technologies, such as HTML pages or Java™ Swing components. Proposed by Elkoutbi et al. [3], UML collaboration diagrams annotated with UI information can be transformed into UI prototypes. In our work, we provide an approach that generates UIs from business processes modeled by business analysts.

Domain-specific languages (DSL) provide high-level languages for code generation in certain problem domains [8]. For example, business process specifications, such as BPEL (Business Process Execution Language) are widely used to integrate Web services using the standards of the service-oriented architecture [9].

In our research, we infer the information embedded in business process specifications to generate UIs for applications.

3 Model-Based UI Generation Framework

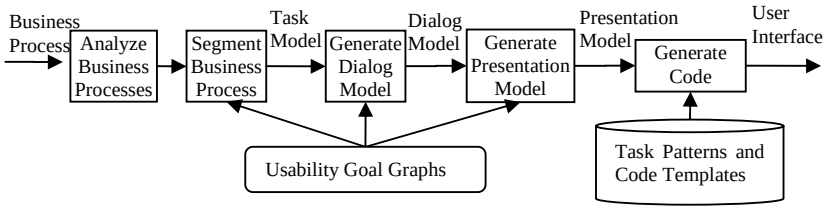


Figure 1: UI generation framework

To reduce the development and maintenance efforts of a UI and its corresponding business process, our UI generation framework uses model transformation techniques that analyze the content of business processes (e.g., tasks, data, and control flows) and produce the source code of an application. Figure 1 illustrates the overall steps in our UI generation framework. However, business processes usually do not contain enough information to automatically generate code. For example, a business process does not describe the threading model or the database schemas to be used in the implementation. We generate intermediate models with increasing details towards the final UI code. The intermediate models include task models, dialog models, and presentation models. Task models recognize the structural and temporal information and describe how roles execute various tasks defined in business processes.

To derive such functional requirements, we must analyze business processes and examine task annotations that describe the functionality of a role (i.e., the person who performs a task). The challenge lies in the fact that the granularity of tasks, expressed in business processes, varies considerably. For example, primitive tasks, such as *"Create an order"* and *"Add a new category"* represent the lowest level of detail. Other tasks may convey functional requirements that can be composed by a sequence of primitive tasks. Instead of considering every task as delivering individual functional requirements, we analyze data dependencies among tasks and data coupling/cohesion in the business processes to group tasks into functional segments. We propose a set of segmentation rules that can group relevant tasks and data into meaningful functional segments. A task model consists of a collection of functional segments. Furthermore, we map one segment into one dialog that specifies the interaction between users and applications.

A dialog model defines the flows (i.e., transitions) of UI windows. We distinguish two types of windows in a dialog model: task windows and support windows. A task window maps to a segment in the task model. A task window allows a user to interact with an application in order to accomplish the tasks in a segment. To improve the user experience, we need to provide more precise information (i.e., input data or output data) about a task. In this way, a business user can understand the functionality of a

task, verify the correctness of their operations once the task is carried out, and easily access auxiliary information for the fulfillment of a task. For example, a picture is a required input to the *"Add product attributes"* task, and a picture of a product may be added into a product catalog after performing this task. A user who is performing the *"Add product attributes"* task would find it useful to be able to preview a picture of the product before inserting it into a product catalog. It may be valuable for the user to be able to verify the product description after the product description is created in a catalog. Support windows provide auxiliary data operations inferred from business processes and assist users performing tasks in the task windows. Window transitions are generated based on the control flows between segments in a task model.

In the presentation model, a dialog is associated with a set of abstract widgets and implements task windows and support windows. An abstract widget describes the generic properties (e.g., title, size, and alignment) of a concrete widget in a specific implementation platform. General speaking, a task can be realized by a set of abstract widgets. For example, a *"Find product"* task can be implemented by a text field that allows a user to enter searching criteria, and a table that displays the search result. To improve the usability of the generated UI, each task or group of tasks is associated with a UI design pattern (e.g., find pattern, output display pattern, and shopping cart pattern). The layout of each window is guided by various UI best practices.

4 Models and Their Transformations

In this section, we introduce the background knowledge about our model-based UI generation. We illustrate the concepts of task models, dialog models, presentation models and transformation rules using examples.

4.1 Analysis of Business Processes

BP = <Task+, Connector* >
Task = <Name, Role, InputConnector*, OutputConnector*>
Connector = <SourceTask, TargetTask, [DataItem], [Condition]>
DataItem = < DataAttr+>
DataAttr = < Name, Type, DefaultValue >
+ means that the occurrence of an element in a tuple is one or more
* means that the occurrence of an element in a tuple is zero or more
[ ] means the this element is optional

Figure 2: The definition of business processes

Our UI generation approach starts by analyzing business processes in order to extract functional requirements and usability information from them. As illustrated in Figure 2, we model a business process in terms of tasks and connectors. Tasks (e.g., *"Find product"*) describe activities that a user needs to perform in order to achieve a business goal. Connectors define the control flow and data flow of a business process. As depicted in Figure 4, an example business process is composed of 9 tasks and 8

connectors. More specifically, a task has a name, a role, zero or more input connectors, and zero or more output connectors. The role specifies who is allowed to perform the task. Input connectors and output connectors describe the input data and the output data exchanged between tasks. A connector links a source task with a target task along with data flows and conditions. For example, as shown in Figure 4, a connector, < *"Add to cart", "Enter quantity", Shopping cart, null*> links its source task *"Add to cart"* and its target task *"Enter quantity"* along with a data item *Shopping cart* that is created by the *"Add to cart"* task and serves as an input to the *"Enter quantity"* task. No data flow conditions are specified in this example. There are three types of control structures in business processes: sequence, decision, and loop. Tasks in a sequence structure will be performed continuously. A decision structure defines more than one possible execution path. Each path has a pre-condition. Once the precondition of a path is satisfied, the tasks in the path are executed. Tasks in a loop structure will be performed multiple times. A loop may have a condition to decide when it stops. Moreover, a data item is composed of a set of data attributes, such as name and type. The data attributes in a business process is specified in an information model that defines the relations among the data items in business processes. Each data item is considered as a class in the information model. The attributes of the data item are interpreted as fields in the corresponding class. A data attribute can be a primitive type (e.g., a string and an integer) or a reference to other data item. For example, the data attribute *billingAddress* in *Payment* data item is an instance of a type *Address*. Moreover, a data attribute may have a default value.

4.2 Task Model

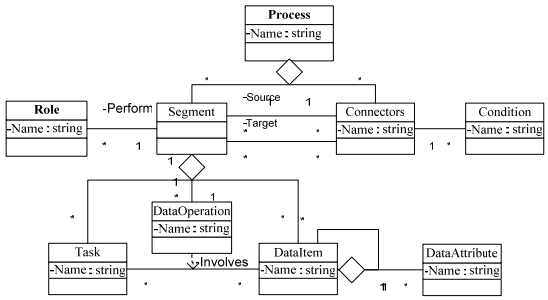


Figure 3: The meta-model of the task model

In the design of UIs, limiting the number of widgets in a window can improve the efficiency of the interactions between users and computers because users can concentrate on performing one task at a time. Elkoutbi et al. suggest a threshold of 20 widgets in a window [3]. As aforementioned, a task can be implemented by a group of widgets. Our aim is to identify the relevant tasks that need to be performed together in the same window, and to limit the number of widgets in a window. We propose several heuristic rules that identify the relevance between tasks and segment tasks by analyzing control and data flows in the business processes. Intuitively,

grouping relevant tasks divides a business process into a set of smaller segments of task groups. Each segment is mapped to a window in the generated UI. The segments are linked by the connectors between segments in a business process. The meta-model of task models is depicted in Figure 3. A business process consists of a set of *Segments* and *Connectors*. Each Segment encapsulates tasks and data items. In the following subsection, we discuss our heuristics for dividing a business process.

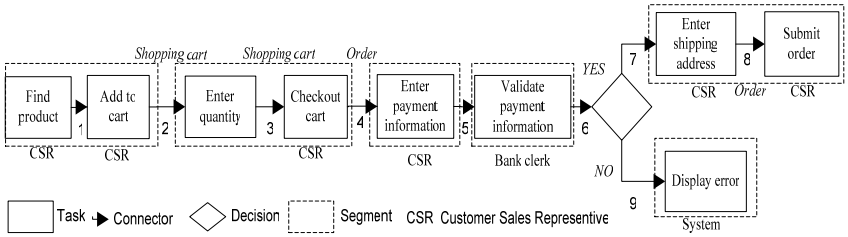


Figure 4: An example business process

4.2.1 Role Rule

$$B_1=\{c|c.SourceTask.Role \neq c.TargetTask.Role\}$$

where *c* is a connector, *c.SourceTask* and *c.TargetTask* describe *c*'s source task and *c*'s target task respectively (Figure 2). *c.SourceTask.Role* and *c.TargetTask.Role* refer to the roles of these two tasks, respectively.

We want to provide a personalized UI for different roles in a business process. A personalized UI provides the necessary functional features and UI widgets with respect to individual roles. A user can focus on their own work and easily select the widgets from fewer widget sets in the UI. We derive a role rule that divides a business process into a few groups as specified in B_1. Each group of tasks is performed by the same role. We identify a set of connectors that link the source tasks of connectors and target tasks of the connectors, performed by different roles. Furthermore, connectors in B_1 divide a business process into *Segments*. Each *Segment* contains the task and related data items performed by one role. For example, as illustrated in Figure 4, the business process is segmented by connectors {5, 6, 7, 9} by applying the role rule.

4.2.2 Primitive Task Rule

$$B_2=\{c|c.SourceTask.IsPrimitiveTask()\}$$

where *c* is a connector, *IsPrimitiveTask()* can identify whether a task is primitive or not

Primitive tasks, such as *"Add to cart"* and *"Submit order"*, describe the lowest level of details in a business process. Such tasks are usually implemented as button widgets and used in combination with other tasks that share the same UI window. For example, when a product is found, the product can be selected from a list of products, and added to a shopping cart. To improve the efficiency of the UI, the *"Add to cart"* task is often combined with the *"Find product"* task and implemented in one window, as illustrated in Figure 4. We identify primitive tasks using naming conventions. For

example, we look for tasks with a name containing *"submit"*, *"save"*, *"add",* and *"delete"*. A primitive task alone does not produce a segment in a business process.

4.2.3 Manual Task Rule

$$B_3=\{c|c.SourceTask.IsManualTask() \oplus c.TargetTask.IsManualTask()\}$$

where *c* is a connector, *IsManualTask()* checks whether a task is a manual task or not

Manual tasks, such as sending surface mail, are manually accomplished by humans without using the e-commerce applications. Therefore, no UI windows are required for performing the manual tasks. The manual task rule is defined by B_3, and used to exclude the manual tasks from the rest of the tasks in a business process. Therefore, we can avoid generating unnecessary widgets or windows for manual tasks in UIs. In B_3, *IsManualTask()* is used to identify a manual task, an attribute that is usually specified in the properties of the task. If either the source task of a connector or the target task of a connector is a manual task, the connector divides the business process into segments by removing the manual task from the business process.

4.2.4 Optional Task Rule

$$B_4=\{c|c.SourceTask.IsOptionalTask() \oplus c.TargetTask.IsOptionalTask()\}$$

where *c* is a connector, *IsOptionalTask()* identify whether a task is a optional task or not

To improve their work efficiency, users would rather click fewer buttons. In this case, the developers may choose to provide default values or settings for UI widgets, such as combo boxes and text boxes. We use the default values specified in the attributes of a data item, which is defined as an output of a task, to provide the default values for the widgets that are used to implement the task. Such a task is considered an optional task to perform only if the user would like to change the default values. For example, the payment by credit card is defined as the default payment method. The task *"Select payment method"* can be skipped. Optional tasks separate the business processes into segments. This rule can improve efficiency because users perform optional tasks only when they cannot use default values. We use the equation B_4 to identify connectors that link to optional tasks. If a task is an optional task, then *IsOptionalTask()* returns true. If either the source task or the target task of a connector is an optional task, such a connector breaks the business process into segments.

4.2.5 Branch Rule

$$B_5=\{c|NumberofInputConnectors(c.SourceTask)>1 \vee NumberOfOutputConnectors(c.TargetTask)>1\}$$

where *c* is a connector, *NumberofInputConnectors ()* and *NumberOfOutputConnectors ()* count the number of input connectors and the number of output connectors of a task.

If a task has more than one outgoing connector, this structure indicates alternative branches or parallel paths. We include all the tasks in one branch as a segment. For

example, the business process in Figure 4 is divided into three segments using the branch rule by connectors {6, 7, 9}.

4.2.6 Data Sharing Rule

$$B6=\{c|(c.SourceTask.InputConnectors\neq null \wedge ic \in c.SourceTask.InputConnectors()) \wedge ic.DataItem() \neq c.DataItem()\}$$

where *c* is a connector, *ic* is one of the input connectors of the source task of connector *c; c.DataItem()* returns the data item associated with connector *c.*

If a sequence of tasks operates on the same data item, these tasks share the same data information. In the UI design, it would be inefficient if a user had to enter the same information multiple times when performing different tasks. This data sharing rule improves the efficiency of the UI design by grouping the tasks and their shared data item in a segment. As specified in B_6, if the data item of the connector *c* and the data item of the connector *ic* are not equivalent, *c* is a connector that divides the business process into different segments. For example as shown in Figure 4, two segments are identified using the data sharing rule. The *"Enter quantity"* task and the *"Checkout cart"* task are included in the same segment since both tasks share the data item, *Shopping cart*. The *"Enter payment information"* task and the *"Submit order"* task share the data item, *"Order"* and are grouped into one segment.

4.3 Application of Rules

Each rule has preconditions that specify the context for applying the rule. For example, the branch rule would not be applied for sequentially ordered tasks. In the process of dividing a business process into a collection of segments in a task model, we identify an applicable set of rules using the preconditions of each rule. The role rule, data sharing rule, branch rule, manual task rule, and optional task are independent from each other. Therefore, the result of the segments is independent from the order of applying these rules. The primitive tasks, such as the "*Add to cart*" and "*Submit order*" tasks are not associated with a separate window. As the result, the primitive task rule identifies primitive tasks, and merges the identified primitive task with the prior task. In this case, the primitive task and their prior task are treated as one merged task. Therefore, we apply the primitive task rule before any other rules. Figure 4 illustrates a segmentation example. The primitive task rule is applied before any other rules and it identifies the connector set B_2. Moreover, the role rule, branch rule, and data sharing rule are applied independently in any orders. These will identify the connector set ($B_1 \cup B_5 \cup B_6$). At last, we derive six connectors that divide the business process into segments (i.e., $B_2 \cup (B_1 \cup B_5 \cup B_6) = \{2, 4, 5, 6, 7, 9\}$).

4.4 Dialog Model

The dialog model is composed of a set of linked windows, as illustrated in Figure 5. Each window is generated from a segment in the task model. Window transitions are

generated from connectors in the task model. In the dialog model, task windows provide widgets that allow users to interact with the application in order to fulfill tasks following predefined orders as specified in the business processes. The Support windows provide contextual support that assists users in performing related tasks. A task window contains task and data operations. A support window can only include data operations inferred from data items. An example of data operations can be viewing the product specification before performing the "*Add to cart*" task. To prevent users from making errors when performing tasks, a window (i.e., editor) can be only editable when a task requires input from the user. Otherwise, we set the window as a viewer that is used for displaying information.

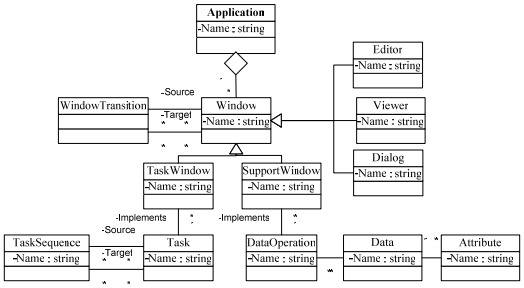


Figure 5: The meta-model of the dialog model

Our rules for transforming task models to dialog models are inspired by [14, 15], which derive navigation structures and window structures from task structures. Our window transactions are directly derived from the connectors in the task models. To generate window structures, we have designed the following three rules to transform segments to windows.

Single Window Rule:

$S \overset{T}{\Rightarrow} W$, where S is a segment, W is a window and $\overset{T}{\Rightarrow}$ is a transformation

We transform all tasks and data operations in one segment into one window. This rule results in a UI that is ideal for expert users, since the users can complete tasks using fewer clicks. But the UI might be difficult for novice users, who have to learn and remember the functionality and order of use of all the widgets in one window.

Data Window Rule:

$S \overset{T}{\Rightarrow} W_T + \sum_{i}^{N} W_{Di}$, where S is a segment, W_T is the task window for the segment S, W_{Di} is the window of the data item i, and N is the number of data items of a segment

This rule transforms all tasks in one segment in a task window and the data operations related to one data item into a support window. In this case, support windows can be

reused in many other segments that contain data operations for the same data item. This rule permits us to reduce the number of tasks and data operations implemented in one window. Moreover, users' knowledge about a support window can be reused in other contexts when the same data items recur.

Functional Window Rule:

$S \overset{T}{\Rightarrow} W_T + W_S$, where S is a segment, W_T is the task window for the segment S, and W_S is the support window for the segment S.

This rule transforms all tasks in a segment into a window and all data operations in the segment into a support window. This rule decreases the number of tasks and data operations in a window. The generated UI is easier to learn than the one generated using the single window rule. However, it may be more difficult to learn than the one generated using the data window rule. This rule is a trade-off between the single window rule and the data window rule.

4.5 Presentation Model

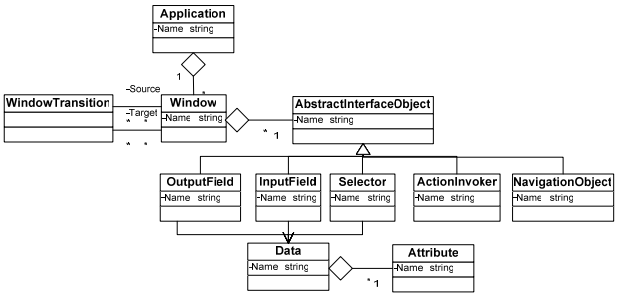


Figure 6: The meta-model of the presentation model

The presentation model transforms the dialog model into a collection of platform-independent abstract interface objects. As illustrated in Figure 6, a window in the presentation model is associated with a collection of abstract interface objects, such as input fields, output fields, selectors and action invokers. The generation of the presentation model is accomplished by matching tasks with task patterns, which are collections of widely used operations in the design of UIs, such as find operation, display contents operation. Two well-known collections of UI task patterns are summarized in [10, 11]. For example, Figure 7 shows the structure of a search task pattern, adapted from [12, 13]. Specifically, a search task can be decomposed into two sub-tasks including entering search criteria and displaying search results. Furthermore, an input field (e.g., text field) and a search action invoker (e.g., button) allow a user to enter search criteria. Similarly, an output field (e.g., a table) and search action invoker (e.g., button) display the returned result. We use naming similarities in order to match tasks in the business processes with task patterns used in the UI designs. The

screenshot for our generated UI after performing the search task pattern on the "*Find product*" task is shown on the right side of Figure 7.

To transform a platform-independent presentation model into a platform-specific concrete UI implementation, a set of platform-specific code templates are used to implement abstract interface objects. The selection of a concrete widget is determined by the tasks and data items specified in the business process. For example, when a task generates output data, the window that implements this task is set to be an editor, for the reason that a user can modify the intermediate result when performing the task. When a data item, such as a *Picture* of a *Product*, is used as an input to a task, the content of this data item can be previewed using a viewer assuming that we can't modify the content of the input data item. We map the types of the data attributes of a data item with different widgets that are used to display the context of a data item. For example, the data item, *Product*, contains string typed attributes, such as *Product ID*, *Product name,* and *Product type*. The string typed data attributes are interpreted into text fields, as illustrated in Figure 7. A data item that can appear multiple times (e.g., an array of resulting products), is converted into a table, as shown in Figure 7.

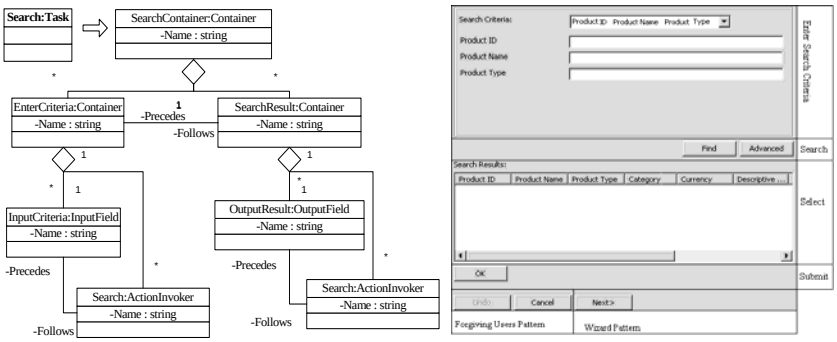


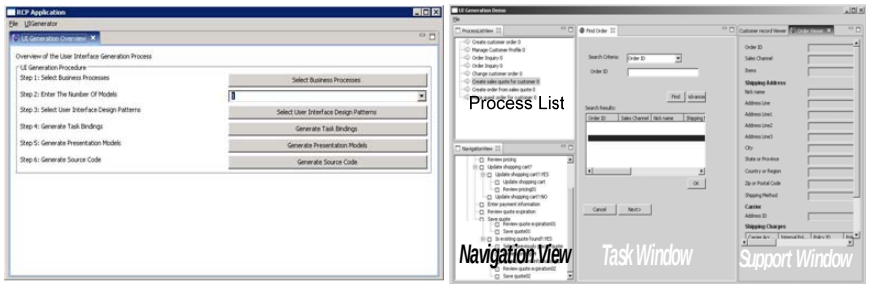
Figure 7: The structure of search task pattern and an example implementation

Moreover, we incorporate several UI design principles during the generation of the UI. For example as shown in Figure 7, three buttons (i.e., *"Undo", "Cancel",* and *"Next"*) at the bottom are generated to improve the usability of this window. The *"Undo"* button and the *"Cancel"* button allow user to correct their mistakes by adapting the *"Forgiving users pattern"* usability pattern [10, 11]. The *"Next"* button is created using the *"Wizard"* pattern [10, 11], which guides users to perform tasks in a step-by-step fashion. Users can use this button to transition to the next page to perform the next task.

5 Case Studies

We designed and developed a prototype tool that can automatically generate UIs from business processes. The screenshot for our UI generation environment is illustrated in Figure 8.a. Our prototype tool parses the business processes that are modeled in

IBM® WebSphere® Business Modeler and stored in XML documents. For the business process modeled using other modelers, a separate parser needs to be developed in order to handle a particular business process specification language. In our UI generation environment, multiple UIs can be generated by applying different UI design principles on the task models, dialog models, and presentation models. Eventually, we allow developers to specify the code templates for implementing abstract UI objects defined in the presentation model.



(a) UI Generation Environement (b) Generated UI Testing Environment

Figure 8: Screenshots of our prototype UI generation tool

Moreover, our generated UIs can be previewed in a test environment, as illustrated in Figure 8.b. The process list view allows developers to access the UI generated from the selected business process. Once a UI for a business process is selected, the structure of the business process is displayed in the navigation view. The task window of the initial task is launched in the area of the task window. The data operations of the task are initiated in the area of the support window. In this screenshot, we applied the data window rule to generate separate windows for each data items and their corresponding data operations.

In the following subsections, we report on the following studies:

1) Evaluate the usefulness of the business process segmentation rules. We track the frequency of the application of each rule in the process of generating task models from business processes.
2) Evaluate the usability of the generated UI.

5.1. Application of Rules

We performed an experiment to count the frequency of the rule application during model transformations. We selected 10 business processes from an existing call center application and counted the frequency of the application of each rule. The results are listed in Table 1. Role rules are applied in each business process, since more than one role participates in the studied business processes. Manual task rules, optional task rules and branch rules are applied in some of the business processes, since not all the business processes contain manual tasks, the data items with default values, and decision paths. As shown in Table 1, data sharing rule is frequently used to merge tasks with data dependencies.

Table 1: Frequency of rule application

Business process	Role rule	Manual task rule	Optional task rule	Branch rule	Data sharing rule
1	1	1	0	0	3
2	1	0	0	0	2
3	1	0	0	1	6
4	1	1	3	2	11
5	1	1	3	1	5
6	1	1	2	3	5
7	1	0	0	1	1
8	1	1	0	0	2
9	1	1	0	0	4
10	1	1	0	0	4

5.2. Usability Evaluation

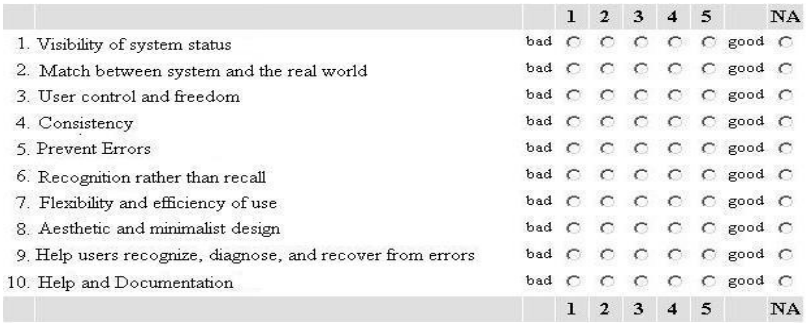


Figure 9: The Nielsen's ten usability heuristic questionnaire [7]

We guide the generation of UIs using UI design principles. To evaluate the usability of the generated UI, we use a standard questionnaire proposed by Chin, Nielsen, and Lewis et al. [7], as illustrated in Figure 9.

We generate two versions of the UI following different UI design principles. One version has high learnability and the other one has high efficiency. For example, high learnability can be achieved by giving more guidance to users. We generate navigation buttons for each window. High efficiency is achieved by grouping the widgets for accomplishing the tasks in one segment into one window. We invite 5 usability researchers to evaluate these generated UIs. The evaluators walked twice through the two versions of UIs of the 10 business processes. Then they evaluated the generated UIs using the questionnaire shown in Figure 9. Each question is scored from 1(bad) to 5(good). If the feature corresponding to a particular question is not available, the score is marked NA. The evaluation results are listed in Table 2. All scores for Q10 are NA, for the reason that we generate UIs without considering context-sensitive help and documentation. Other questions are marked as good or

above average. Hence, we believe that the usability of the generated UI is acceptable for the professional usability researchers.

Table 2: Usability evaluation result

Question	E1	E2	E3	E4	E5
Q1	5	5	5	5	5
Q2	5	5	5	5	5
Q3	4	5	5	4	5
Q4	5	5	5	5	5
Q5	5	4	4	5	4
Q6	5	5	5	5	5
Q7	5	5	5	5	5
Q8	5	5	4	5	5
Q9	5	4	5	4	5
Q10	NA	NA	NA	NA	NA

6 Conclusion and Future Work

In this paper, we present our approach for automatically generating UIs from business processes. Business processes are used as business requirement models that capture business knowledge. The task models are directly derived from business processes. The UIs are generated using two other intermediate models: the dialog models and the presentation models with increasing levels of details. To ensure that the generated UIs are easy to use and learn, the transformations between models are guided by UI design principles and task patterns. As a result, the generated UI has strong usability supports such as consistent look and feel, and transition guidance. Moreover, any changes to business processes can be automatically propagated to the UIs by regenerating the code using our prototype tool. So, it is easy for developers to integrate their feedback to the generation process and to fine-tune their design.

In the future, we plan to extend our prototype tool to allow users to integrate their own task patterns and UI design principles into the generation process so that they can reuse their existing knowledge to generate UI following their existing style. We will improve the business process analysis technique by identifying additional contextual information from business processes.

Acknowledgement

This research is sponsored by IBM Canada, Centers for Advanced Studies (CAS), and National Sciences and Engineering Council (NSERC). We would like to thank Mr. Qi Zhang at the University of Waterloo and Mr. Tack Tong at the IBM Canada Toronto Laboratory for their suggestions on this work.

Trademarks

IBM and WebSphere are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Disclaimer

The opinions expressed in this paper are those of the authors.

References

1. Qi Zhang, Rongchao Chen, and Ying Zou: Reengineering User Interfaces of E-Commerce Applications Using Business Processes. ICSM (2006): Pages 428-437
2. Anura Guruge, “Corporate Portals Empowered with XML and Web Services”, Elsevier Science (USA) (2003)
3. Mohammed Elkoutbi and Rudolf K. Keller: User Interface Prototyping Based on UML Scenarios and High-Level Petri Nets. In Proceedings of ICATPN (2000): LNCS 1825, Pages 166–186
4. Douglas C. Schmidt: Guest Editor's Introduction: Model-Driven Engineering. IEEE Computer (2006): Volume 39, Issue 2
5. Jeffrey Nichols and Andrew Faulring: Automatic Interface Generation and Future User Interface Tools. In Proceedings of the Workshop on the Future of User Interface Design Tools at CHI (2005)
6. Angel R. Puerta: A Model-Based Interface Development Environment. IEEE Software (1997): Volume 14 , Issue 4, Pages 41-47
7. Jakob Nielsen: Nielsen's Heuristic Evaluation. Usability Engineering. Academic Press (1993): Chapter 5, Page 115.
8. Uwe Zdun: Concepts for model-driven design and evolution of domain-specific languages. In Proceedings of the International Workshop on Software Factories at OOPSLA USA(2005): Pages 1-6
9. Frank Leymann, Dieter Roller, and Marc-Thomas Schmidt: Web services and business process management. IBM Systems Journal (2002): Volume 41, Number 2
10. Martijn van Welie: Patterns in Interaction Design. http://www.welie.com/
11. Jennifer Tidwell: Designing User interfaces. http://designinginterfaces.com/
12. Sinnig, D., Gaffar, A., Reichart, D., Forbrig, P., Seffah, A.: Patterns in Model-Based Engineering, In Proceedings of CADUI (2004): Pages 197-210
13. Fabio Paternò, Model-Based Design and Evaluation of Interactive Application. Springer Verlag (1999): ISBN 1-85233-155-0
14. Bradley T. Vander Zanden and Brad A. Myers: Automatic, Look-and-Feel Independent Dialog Creation for Graphical User Inter-faces. In Proceedings of SIGCHI (1990): Pages 27-34.
15. Jean Vanderdonckt: Knowledge-Based Systems for Automated User Interface Generation: the TRIDENT Experience. In Proceedings of the CHI Workshop on Knowledge-Based Support for the User Interface Design Process (1995)