Developing Navigational User Interfaces Using Business Processes

Qi Zhang[1], Rongchao Chen[2] and Ying Zou[3],
Department of Electrical and Computer Engineering
Queen's University
Kingston, Ontario, Canada
{3qz[1], 4rcc[2]} @qlink.queensu.ca, ying.zou@queensu.ca[3]

Abstract

Business users are often overwhelmed by the enormous functionality available in business applications, such as e-commerce applications, necessary to accomplish activities required by business processes. The business users are required to take continual training to use e-commerce applications, since the user interfaces of these e-commerce applications are frequently updated to reflect the continuous evolution of the underlying business processes. Business processes describe the functionality of e-commerce applications from the perspective of business users. In this position paper, we utilize the knowledge embedded in business processes and generate navigational user interface components that implement business tasks defined in business processes. We aim to provide contextual information and guide users to fulfill business tasks step by step in order to achieve business objectives.

1. Introduction

A business process is a sequence of tasks which need to be carried out to achieve business objectives for the organization. For example, a book purchasing process may consist of several tasks, such as selecting a book from the catalog, using a credit card for the payment, and printing out a receipt. A workflow provides specifications for describing business tasks, roles, data and resources involved in a business process. Business applications are designed to support business tasks specified in business processes. To achieve the escalating requirements from business users, business applications, such as e-commerce applications, have gradually evolved and provide sophisticated functional features in user interfaces. However, such rich features are often not obvious for users to navigate in the user interfaces (UIs). As a result, business users, especially novice business users may struggle in deciding where to start or where to go next after a result is received from a particular toolbar in the user interface. One of the reasons is that the user interfaces of business applications are often designed from the perspective of developers, but not from usability point of views of business users. Therefore, business users need to take continual training in order to be competent to perform business process activities using these business applications. These problems result in increase in operation cost and decrease in business performance. Therefore, a more systematic approach is required to design these business applications to ensure business users can carry out business activities in these applications more efficiently and effectively.

In this paper, we propose an approach for developing business process driven user interface. We leverage the knowledge in the business processes to improve the usability of user interfaces of business applications. We automatically generate navigational UI components which indicate the progress of a business process and automatically prompt the next task for the user to accomplish. Our aim is to guide the user to operate on an appropriate existing UI component that is developed to fulfill a business task. To integrate the generated navigational user interface components with existing UI components, we define the linkages between the two. Furthermore, we adapt contextual information embedded in business processes to dynamically provide the UI components relevant only to currently progressing business processes.

2. An Approach for Developing Business Process Driven User Interfaces

Business processes describe the functionality of a business application using a set of task specifications

in workflows. Moreover, a business process defines a possible processing order in which business users interact with the user interface components that fulfill business tasks. A business process also describes the contextual data and resources that assist the fulfillment of each business task. As a result, the knowledge in the business process can be leveraged and served as the initial design requirement for the user interfaces of the business application.

Typically, an UI component is represented as a dialog, an editor window or a view window in a business application. Each UI component may implement one or many tasks. This association is known as the binding between UI components and the tasks in a workflow [2][6]. As an example, a typical *Purchase order* process for a retail sales application is shown in figure 1. The *Enter customer information* task can be completed in an order editor window. The *Add product to order* task and the *Enter payment information* task can be fulfilled in the Select product dialog and payment dialog, and finally the *Submit order* task needs to be done in the same order editor. Essentially, the business user completes the business process by navigating to different UI components and performs tasks in each component. In our approach, we utilize this information to ease the uses of a user interface.

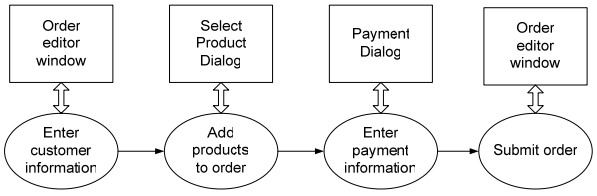

Figure 1: Binding tasks to user interface components for a Purchase order process

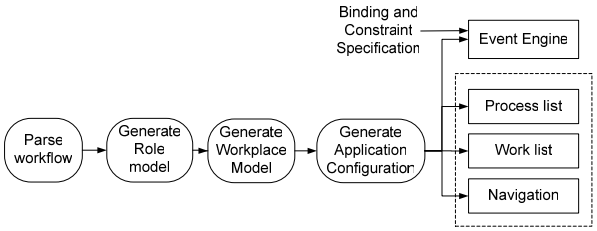

Figure 2: A framework for developing business process driven user interfaces

2.1 A Framework for Designing Business Process Driven User Interfaces

Our framework for developing business process driven user interfaces was detailed in our previous work [6]. The structure of our framework is shown in Figure 2. Our framework first parses workflows. It then generates the role model which describes tasks required for each business role (e.g., sales representative) to perform. The role model is further transformed into a workplace model, which defines the basic layout and content of the user interface components. In our framework, we automatically generate three UI components that facilitate users to access the tasks, including *Process List*, *Work list* and the *Navigation.* The *process list* allows the user to instantiate a workflow instance by selecting a process name in the list. The *Work list* is used to show the tasks that need to be completed by the user. The *Navigation* is used to display the flow structure as well the status of the workflow instance. Finally, the workplace model is converted into an application configuration, which is used to configure the layout and content of UI complements implemented by a particular UI technologies, such as the Web portals [8]and Eclipse Rich client platform [9]. The configuration is processed by the *event engine* to configure navigational UI components, including the *Process List*, *Work list* and the *Navigation* at run time. Configurations contain bindings and constraint specifications, which allows for integrating the navigational UI components with the existing business application. By incorporating business process flow information in the user interface, we provide a user-guidance environment for user to perform daily work. Detailed operations of each component are mentioned in the next section.

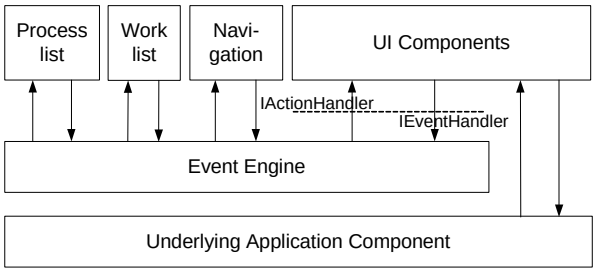

Figure 3: Architecture of Workflow-driven Business Application

2.2 Architecture for Event Engine

The architecture of our business process driven user interface for business applications is shown in figure 3. The *Process list*, the *Work list* and the *Navigation* components communicate with the event engine to obtain the run time workflow instance information. To solve the consistency issue between the user interfaces and its underlying business processes, we created a binding between the two, which includes an

event interface, known as *IEventHandler*, and an action interface, known as *IEventHandler*, as illustrated in Figure 4. The *IEventHandler* is used by the UI components to send events to the *event engine*. For example, a UI component can send an UI event that indicates the start or end of a task. The *IActionHandler* interface is used by the *event engine* to drive the UI components. The current actions supported include open, close, highlight and de-highlight UI components. In a typical execution sequence, when a task needs to be executed, the *event engine* automatically opens (or shows) the UI component to the user. During an instance switch, the event engine de-highlights the UI components bound to a UI component, and highlights the UI components bound to the new workflow instance. However, in certain cases, it is not desirable for event engine to automatically open the UI component, since the user may want to verify the result of the previous step. In this case, we allow the user to open the UI component manually.

When the user is working in one UI component, it sends events to indicate the start and end of the tasks. In addition, tasks can be aborted. For example, the user can select the Cancel button in a dialog to undo the changes in the dialog. In this case, a terminate event is also sent by the UI component. Further, some task may require no user interaction. For example UI component may provide a default value so the user does not need to perform any actions in the UI. Even more, some tasks can be done repeatedly. An example is the *Add products to order* task shown in figure 1. This task can be carried out multiple times, depending on the customer's decision. Moreover, some tasks are also restrained by additional constraints, such as application data and the execution sequence. These properties and constraints are captured in a constraint configuration. These constraints can be provided by the software developers based on their design of the business application.

```
public interface IActionHandler {
  public void OpenUIComp(String ApplicationName,
         Object data);
  public void HighlightUIComp (Object application);
  public void DehighlightUIComp (Object application);
  public void CloseUIComp (Object application);
}

public interface IEventHandler {
  public void fireWorkflowEvent(String eventName,
         Object[] data);
}
```

Figure 4: Definitions of the *IActionHander* and the *IEventHandler* interface

The advantage of our approach is that it supports multiple workflow instances. For example, two *Purchase order* process can run at the same time, in two different order editors. The *event engine* distinguishes run-time workflow instances based a set of unique identifiers. These identifiers could be the UI component itself, or the data being processed by the UI components. When UI event occurs, the event also sends the unique identifiers along with the event (represented by `Object[] data` in figure 4). Given the uniqueness of the identifiers, the *event engine* always knows which workflow instance is being processed by the user.

```
<Configuration>
<ProcessBinding ProcessName="Purchase Order">
  <TaskBinding TaskName="Enter customer information"
         Optional="false" Manual="false">
<Application Name="OrderEditor" Invoke="true"
         Close="false" />
  </TaskBinding>
  <TaskBinding TaskName="Add products to order"
         Optional="false" Manual="false">
    <Application Name="SelectProductDialog" Invoke="false"
         Close="false" />
  </TaskBinding>
  <TaskBinding TaskName="Enter payment information"
         Optional="false" Manual="false">
    <Application Name="PaymentDialog" Invoke="false"
         Close="false" />
  </TaskBinding>
  <TaskBinding TaskName="Submit order" Optional="false"
         Manual="false">
    <Application Name="OrderEditor" Invoke="false"
         Close="false" />
  </TaskBinding>
</ProcessBinding>
<TaskConstraint TaskName="Enter customer information">
  <TaskPrecondition>
    <EventConstraint
         EventName="Start Enter customer information" />
  </TaskPrecondition>
  <TaskPostcondition>
    <EventConstraint
         EventName="End Enter customer information" />
  </TaskPostcondition>
  <TaskCancelcondition>
    <EventConstraint EventName="Order editor closed" />
  </TaskCancelcondition>
</TaskConstraint>
    ...
</TaskConstraint>
<Configuration>
```

Figure 5: Binding and constraint specification for the Purchase order process

Another advantage of our framework is that since the event engine has the knowledge of the progress of each workflow instance, the event engine can provide support in the business application by displaying related information or tools related to the current task. When the task starts, the relevant information and tools automatically appears. When the task ends, these UI components are removed. Furthermore, when the user switch instance in the workplace, the supporting tools are automatically displayed to or hidden from the user, based on the context of the current progressing workflow instance.

2.3 Binding and Constraint Format

In this section we elaborate more on the format of our binding and constraint specification. The binding specification is used to specify the linkage between UI component and the tasks in the business process, and to integrate navigational UI components with existing UI components. The constraint specification describes each event produced by the UI components and the mapping between events and the start and end condition of tasks. This provides flexibility for the developers to modify the source code to generate events in the UI components.

As an example, the binding and constraint specification for our example process in figure 1 is shown in figure 5. The first part is the binding specification that defines the binding between UI component and the workflow task. It also contains the properties of the tasks. A task can be manual, optional or repeatable. A manual task is a task which is performed without interacting with the business application. In this case, the task is not bound to any UI components. An optional task is a task which may require no user interaction. A repeatable task is a task that can be done repeatedly. The *Invoke* and *Close* property is used to indicate whether the UI component needs to be opened or closed automatically once the task needs to be executed or completed. The second section is the constraint specification. Each task in the process is associated with pre-conditions, post-conditions and cancel-conditions. Each condition specifies the start, end and abort condition of the task. Each condition is composed of constraints, such as event constraint, data constraint and workflow structural constraints. These constraints allow the event engine to interpret events and update the state of workflow instances correctly.

3. Case Studies

To further examine our process driven user interface framework, we developed a prototype tool in Eclipse, which is an open extensible platform and Integrated Development Environment (IDE) for Java programming. We integrate our framework with a call center application with 100K LOC. This system provides full functionality for a sales representative to interact with customers and sales systems, however, with drawbacks and limitations in terms of usability and consistency with its underlying business processes. For example, novice users will be overwhelmed by its numerous functions provided through its menu items and may get lost during operation since some proceeding buttons are hidden deep inside. The e-commerce application that we used in our research inherits the Eclipse presentation style, which includes its workbench, views, editors and menu items, and is known as Eclipse Rich Client Platform (RCP). Our prototype tool is integrated with the call center application as an Eclipse plug-in. The screenshot of the prototype system is illustrated in Figure 6.

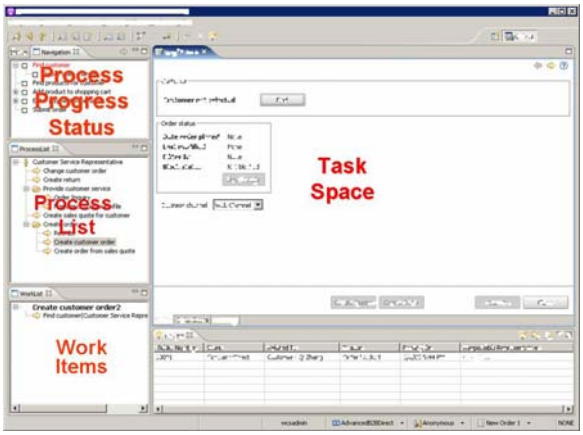

Figure 6: Prototype Process Driven User Interface

Within the original call center sales application, events were inserted into the user interface components classes that are tightly bound to business workflow tasks, such as the creation of an order editor instance which is related to the create order task in business process. There are approximately 30 events in total and they are handled by the event engine within our plug-in project and corresponding actions are performed from the business task that is bound to the event through the pre-defined action interface.

Bindings between business processes and application user interface are achieved by using a configuration file. In order to maintain the integrity fo the original design of UI components, we decided to set the level of granularity in the binding to user interface components such as editors, views and dialogs as comparison with other research [2] that binds tasks to lower levels such as widgets. The template for this binding configuration is automatically generated by parsing through business process and workflow graphs. The developers only need to fill in the user interface component configuration when using it.

Our prototype tool enhances the application's usability and displays underlying business processes information. However, our current implementation still has limitations. Since the business processes are not defined based on the user interface, there is always gap between the two. For instance, multiple business processes can be implemented on one user interface component to save operation time. Thus, the mapping

between business tasks and user interface components can be difficult to identify in some cases.

4. Conclusion

In this paper we proposed an approach for developing navigational user interfaces using business Processes. We aim to provide guidance to facilitate users to fulfill business tasks. In the future, we would like to further refine our approach. We plan to conduct usability studies to test the effectiveness of our approach.

Acknowledgement

This research is sponsored by IBM Canada, Centers for Advanced Studies (CAS), and National Sciences and Engineering Research Council (NSERC).

We would like to thank Jen Hawkins, Bhadri Madapusi, Tack Tong and Ross McKegney for their valuable contribution to this work.

References

[1] "From business reengineering to business process change management: a longitudinal study of trends and practices", *IEEE Transactions on Engineering Management*, Vol. 46 No. 1, pp. 36-46.

[2] Sliski, T.J., Billmers, M.B., Clarke, L.A. and Osterweil, L.J. "An Architecture for Flexible, Evolvable Process-Driven User Guidance Environments" *Proceedings of the Joint 8th European Software Engineering Conference (ESEC 2001) and the 9th ACM Sigsoft Symposium on the Foundations of Software Engineering (FSE 9)*, September 2001, Vienna Austria, pp. 33-43.

[3] Ying Zou, Terence Lau, Kostas Kontogiannis, Tack Tong, Ross Mckegney, "Model Driven Business Process Recovery", *in the Proceedings of the IEEE Working Conference on Reverse Engineering (WCRE)*, Delft, The Netherlands, Nov. 2004.

[4] Becker J., zur Muehlen M., Gille M "Workflow Application Architectures: Classification and Characteristics of Workflow-Based Information Systems", *Workflow Handbook 2002*, Future Strategies, Lighthouse Point, 2002.

[5] D.-A. Manolescu and R. E. Johnson. A micro workflow framework for compositional object-oriented software development. OOPSLA'99 Workshop on the implementation and Application of Object-Oriented Workflow Management Systems II, Nov. 1999.

[6] Qi Zhang, Ying Zou, Tack Tong, Ross MacKegney and Jen Hawkins, "Automated Workplace Design and Reconfiguration for Evolving Business Processes", *to appear in the 15th Centre for Advanced Studies Conference (CASCON)*, Toronto, Canada, November 2005, pp. 262-277.

[7] "Business Process Execution Language for Web Services version 1.1". ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

[8] Corporate Portals Empowered with XML and Web Services, Butterworth-Heinemann Newton, MA, USA, 2002

[9] Eclipse Rich Client Platform. http://www.eclipse.org/rcp/