Verifying Business Processes Extracted from E-Commerce Systems Using Dynamic Analysis

Derek Foo[1], *Jin Guo*[2] *and Ying Zou*[1]

Department of Electrical and Computer Engineering[1]
Queen's University
Kingston, Ontario, Canada
3kcdf@qlink.queensu.ca, ying.zou@queensu.ca

School of Computing[2]
Queen's University
Kingston, Ontario, Canada
guojin@cs.queensu.ca

Abstract

E-commerce systems must react in real-time to user inputs and business rules. For the purpose of re-documentation, static analysis is often adopted to recover business processes implemented in e-commerce systems. However, static analysis fails to recover the complete tasks in business processes due to the dynamic nature of e-commerce systems. To improve the accuracy of recovered business processes, we devise dynamic analysis techniques which trace the execution of processes. We recover usage scenarios from the execution logs and use them to verify the business processes recovered using static analysis. We verify the effectiveness of our proposed approach through a case study on OFBiz e-commerce applications.

1. Introduction

Most e-commerce systems are constructed using three-tier architecture, which includes user interface (UI) tier, business logic tier, and database tier. E-commerce systems are highly dynamic. They react to users' requests in real-time and generate results according to user's selections, business rules and status data stored in databases. For the example of an on-line bookstore, a book can be placed in the shopping cart only if the book in stock. An e-commerce system implements a collection of business processes that describe the operations provided by an organization. Specifically, a business process consists of tasks, control flows, data and participants. A task is a unit of work, which can be executed either automatically in the business logic tier (i.e., back-end components) or require human interactions through UIs. For example, a book purchase business process can contain "selecting a book" task, "select a payment method" task and "buying a book" task. Control flows describe the task execution paths [4], such as sequential, alternative and parallel paths.

Business processes are often optimized to reduce the cost of business operations and improve the quality of services provided by an organization. Business process optimization requires updating the source code to accommodate the modified part of the business processes. When performing code updates, developers need to identify the portion of the code to change. However, locating the code blocks corresponding to a particular feature or change is a difficult process without up-to-date documentation.

In previous research [2][3], we applied static analysis techniques to examine source code and reason over all possible behaviors that might arise during run-time, in order to recover business processes in e-commerce systems. To reduce the complexity of representing the recovered business processes, our previous work [3] represents the business processes in terms of two abstraction levels: high-level and low business processes. High-level business processes give an overview of business operations and contain tasks which require human interactions or which are executed by components in the business logic tier. Low-level business processes contain the details steps performed in the back-end components. Separating complex system structure can provide a clear view of the overall structure of an ecommerce application while hiding processing details. However, static analysis cannot capture the business tasks executed under the conditions determined at run-time. For the example of an on-line bookstore, when a selected book is not available, alternative books can be recommended to the user. However, the business task which recommends alternative books cannot be detected using static analysis. Moreover, we cannot effectively verify the accuracy of the recovered business processes without manually examining the source code.

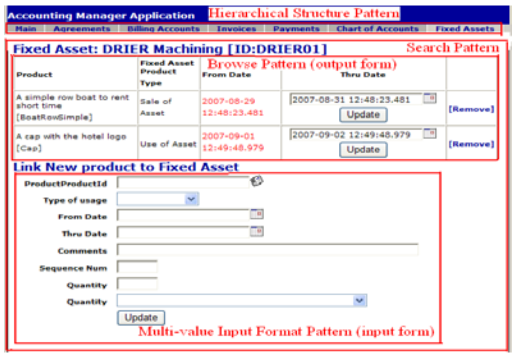
To improve the accuracy of recovered business processes, we aim to integrate static analysis with dynamic analysis for recovering business processes. In this paper, we use dynamic analysis techniques to observe the behaviors of a system at run-time. We also use the result of the dynamic analysis to verify and enhance business processes recovered from static analysis. More specifically, the static analysis identifies business processes from the source code of three tiers of the e-commerce application. As an extension to our previous work [3] we refine our techniques for recovering high-level business processes using UI design patterns, which describe the best practices to implement UI functionalities. The UI design patterns are used to separate multiple business processes implemented in the same UI screens. The dynamic analysis records how a user would normally interact with the e-commerce applications. Usage scenarios are generated to represent the steps that a user needs to perform in order to complete business processes. We record the tasks performed throughout the three tiers when a user interacts with the systems to fulfill business processes. When a user conducts a particular usage scenario, information regarding each step in the scenario is recorded. Since static analysis may not be capable of recovering business processes correctly when run-time information is needed, therefore, we match the tasks from business processes recovered using static analysis and the usage scenarios traced using dynamic analysis. Depending on the results of the matching, we can determine whether a recovered business process is complete or incomplete.

The rest of the paper is organized as follows. Section 2 presents techniques for recovering business processes using UI design patterns. Section 3 presents the techniques for identifying usage scenarios. Section 4 discusses the case studies. Section 5 gives a brief overview of related work. Finally, Section 6 concludes our work and discusses future work.

2. Recovering Business Processes using Static Analysis

We recover business processes from the three tiers of e-commerce systems using static analysis. Recovering business processes from UIs is a challenging task, since the UI is often developed without referring to the underlying business process specifications. The structure of a UI can be complex due to the hyperlinks and hierarchical structures of the widgets (e.g., buttons, tables, and trees in UIs). The hyperlinks can connect one page to other pages with arbitrary orders. Therefore, it is difficult to identify the control flow constructs (e.g., sequence, alternative and parallel) between the functionality delivered in different pages in an e-commerce application. It is also hard to determine the starting point of a business process from the interwoven page links.

The hierarchical structure of a widget complicates the identification of a task with appropriate granularity. For example as shown in Figure 1, a table widget contains multiple cells in different rows and columns. Each cell can further encapsulate tables, hyperlinks, buttons and selection lists. Many possible tasks with different level of granularity can be identified from the UI widgets. For example, the entire table can be considered as a task that displays the result of a search operation. A button widget that triggers a back-end service could be considered as a task with low-level details. However, it is difficult to understand a business process with excessive low-level detailed tasks since a business process is intended to capture high level business operations.


Accounting Manager Application
Hierarchical Structure Pattern
Main
Agreements
Billing Accounts
Invoices
Payments
Chart of Accounts
Fixed Assets
Search Pattern
Browse Pattern (output form)
Update
Multi-value Input Format Pattern (input form)

Fixed Asset: DRIER Machining [ID:DRIER01]

Product	Fixed Asset Product Type	From Date	Thru Date	
A simple row boat to rent short time [BoatRowSimple]	Sale of Asset	2007-08-29 12:48:23.481	2007-08-31 12:48:23.481 Update	[Remove]
A cap with the hotel logo [Cap]	Use of Asset	2007-09-01 12:49:48.979	2007-09-02 12:49:48.979 Update	[Remove]

Link New product to Fixed Asset

ProductProductId	
Type of usage	
From Date	
Thru Date	
Comments	
Sequence Num	
Quantity	
Quantity	

Figure 1. Example UI Page of Sequoia ERP [11]

To overcome the complexity of UI implementation, we use UI design patterns to abstract the structure of the UIs and capture the tasks and controls flows from the structure of pages, and the dependencies among pages. We classify UI design patterns into three categories according to the structure of the UI:

1) Hierarchical structure patterns: are used to organize the overall structure of a UI and group different functionalities of a UI into subsystems. Each subsystem allows a user to work on particular functionality. For example, as shown in Figure 1, the multiple tabs depicted on the top of the screen allow a user to select different subsystems (e.g., agreement, accounting, and catalog) to work on. This tab structure is an instance of the hierarchical structure pattern. The first screen displayed after selecting a subsystem from the tabs suggests the starting point of a business process.
2) Navigation patterns: describe the navigational structure of UI pages. The navigational structure allows a user to accomplish one business process

by navigating through different pages. Generally, a business process can be implemented within one page or can be accomplished across multiple pages. To improve the usability of the UI, different UI navigation patterns are often adopted to guide users through the UI pages. For example, a wizard pattern guides a user to complete tasks in a step by step fashion. We consider the operations fulfilled in the wizard pattern as a business process and we map each operation in the wizard pattern to a task in a business process. The order of each step is converted to a sequential control structure in the business process.

3) Behavioral patterns: characterize functional units delivered in a page. For example as shown in Figure 1, a multi-value input format pattern [10] encapsulates multiple input widgets that take input data from a user; a browse pattern [10] presents a set of ordered data as the output from the back-end components; a search pattern [10] allows a user to specify search criteria, and to review the results of a search. We locate the tasks by identifying the behavior patterns in a UI page.

3. Verifying Business Processes using Dynamic Analysis

The goal of dynamic analysis is to generate a sequence of business tasks which represent the processing steps for fulfilling a business process. A business task, however, can be implemented as code fragments with utility code and other non-business related task in between. Normally, a business task is associated with business data accesses and manipulations. Simply recording all code lines that involve business data would quickly lead to the creation of large log files that is challengint to analyze . Utility classes provide internal basic operations which facilitate the completion of a business process. These non-business logics do not contribute to business processes and hence, should not be logged.

To filter non-business logic, we define criteria to guide the insertion of instrumentation code into the different tiers of an e-commerce application. We define a set of rules to trace the execution of business tasks using code instrumentation in all tiers. Aspect oriented programming can be applied to insert the probes automatically at the location where the rules can be applied. The recorded information from each tier is merged and sorted by access time in order to create a complete usage scenario of a business process.

A potential drawback of dynamic analysis is that it requires a large set of test cases to ensure it covers all execution path of the application. The effort of generating such test case suites makes it impractical for applying dynamic analysis to large software such as e-commerce application. Instead of using dynamic analysis to recover business processes, we apply dynamic analysis to verify the results of static analysis and identify possible mistakes in the business processes generated from static analysis.

3.1 Instrumenting the User Interface Tier

The UI tier is responsible for generating page transitions and collecting user's inputs. This tier is the starting and ending point for all business processes. Upon collecting all necessary information from the user, the UI tier forwards the request of the operation to the business logic tier. We identify two rules for signaling the starting and ending point of a business process.

Rule 1. Each business task is initiated by a user's interaction with the UI. We log each user request as well as the input data for the request.

Rule 2. Upon completion of the user's request of a business task, a new UI screen containing the result of the request is generated. This signifies the end of the business processes and is recorded.

3.2 Instrumenting Business Logic Tier

The business logic tier contains the functional algorithms which process requests from the UI tier using information from the databases. While the other two tiers remain unchanged most of the time, the business logic tier is often updated to reflect changes in the business processes. We identify a preliminary set of rules for instrumenting the business logic tier.

```
1 Public static createPayment(Object paymentType){
2            createPaymentMethod("Credit", null);
3 }
4 Public static createPaymentMethod(Object paymentType, Object paymentDate){
5            ......
6 }
```

Figure 2. Implementation to support previous releases

Rule 1. To provide support to the previous releases of a system, the implementation of a function may contain code which transfers the program control to a newer implementation of the function. As shown in Figure 2, the implementation for *createPayment* from line 1 to 3 contains only an invocation of the newer version of the function, namely, *createPaymentMethod*. Since the program control is being transferred to another method, it is assumed that the method being transferred to contain all business logics. Therefore, the instrumentation should start at the method being invoked. For example as illustrated in Figure 2, the probe for tracing the execution of the function (i.e., a business task) is inserted at line 4.

Rule 2. Identify the starting point of a business task which is completed by invoking multiple methods in source code. In most e-commerce systems, business tasks are declared as services in an interface file and implemented in a different file. By analyzing the service declaration, a precise starting point of the implementation of a business task is identified.

Rule 3. Identify business data which govern the purpose of the business tasks. The business data is passed from the UIs and is used to implement business tasks and enforce business rules that specify the conditions for executing business tasks. To generate a meaningful trace, only the code lines that involve business data validation or modification should be recorded. In later cases, data modification is done by populating variables with business data. If the variable is modified multiple times, we only record its last assignment as this is the final value that would be stored in the database.

3.3 Comparing Results from the Static Analysis and Dynamic Analysis

Users often perform multiple business processes in one session. To detect each business process, we analyze the traces produced from all users' requests made during a session. The usage scenarios can be recovered from the traces. Specifically, a usage scenario captures the tasks that are carried out to react to user's requests. Different from recovering a business process, the control flow constructs (e.g., sequential, alternative and parallel execution paths) are not included in a usage scenario.

To compare the results from both analyses, we match the tasks identified from both techniques in sequential order. There are three possible outcomes through the comparison.

1) If a task is present in the results of both analyses, then the task is verified in the recovered business processes.
2) If a task is present in the dynamic usage scenario of dynamic analysis but not in the recovered business processes from the static analysis, we would treat it as a missed task in which static analysis failed to recover a business task. We complement the static analysis result with the missing task found in dynamic analysis result.
3) If a task is present in static analysis but not in dynamic analysis, we would consider it as a possible misidentified task and we can manually analyze the corresponding code block to determine if it is a business task.

4. Case Study

To demonstrate the effectiveness of our proposed approach, we performed case studies on Sequoia ERP [11], a variant of the Open For Business project (OFBiz). The system is implemented in Java and a proprietary scripting language, called Mini-Language.

We have instrumented the Sequoia by applying the rules discussed in Section 3 on four Sequoia ERP subsystems: catalog, facility, marketing, and workeffort. Using static analysis, we recovered 1) high-level business processes from the scripting code for UI pages and the XML scripting code, 2) low-level business processes from the Java source code. To produce the traces, we recruited an undergraduate student to use the system. The undergraduate student studied the user guides and on-line demos of the system before the experiment.

We developed a prototype tool to recover usage scenarios that have the same starting points as the business processes recovered using static analysis. Such usage scenarios serve as the basis for the verification.

We compare the results by matching the name of tasks and starting points of the process in the results of both analyses. We count the number of missed tasks and misidentified tasks based on the criteria mentioned in section 3.3. We calculate the recall rate and precision using Eqn 1 and Eqn 2.

$$precision = \frac{\#of\ \ identified\ tasks - \#of\ \ misidentified\ \ tasks}{\#of\ \ identified\ tasks} \quad \text{(Eqn 1)}$$

$$recall = \frac{\#of\ \ identified\ \ tasks - \#of\ \ missed\ \ tasks}{\#of\ \ identified\ \ tasks} \quad \text{(Eqn 2)}$$

Table 1 summarizes the high-level business processes recovered using static analysis from the four studied subsystems in Sequoia ERP. We have recovered 116 business processes and 233 unique tasks in total. We calculate the precision and recall for misidentified tasks and missed tasks using Eqn 1 and Eqn 2 respectively. The mis-identified tasks should be removed from the recovered business processes. The result of the comparison is shown in Table 2. Comparing the usage scenarios recovered from the dynamic analysis, we locate 4 business tasks that were missed using static analysis. No misidentified tasks were found. The precision and recall are 100% and 98.3% respectively as shown in the Table 2.

Table 1. High-Level Business Processes from Sequoia

Subsystem	# of Workflows	# of Tasks
catalog	66	133
facility	30	62
marketing	8	10
workeffort	12	28
Total	**116**	**233**

Table 2. Precision and Recall for the Static Analysis

	# of Ident. Task	# of Mis-ident. Task	# of Missed Tasks	Precision	Recall
High-Level Process	233	0	4	100%	98.3%
Low-Level Process	195	46	32	76.4%	83.6%

For the low-level business processes, our static analysis techniques missed 32 tasks and mis-identified 46 tasks. We found the precision for recovering low-level process is 76.4% using static analysis. Upon examining the result, we found that the misidentified tasks were mostly related to debugging statements used in the system. We use the results of the dynamic analysis to add the missed tasks and correct the mis-identified tasks. .

5. Related Work

Static analysis is adopted in previous research [1][2][3] to recover business processes. Huang *et al.* [1] propose an approach which depends on variable classification to recover business rules. Zou *et al.* [2] describe a model-driven business process recovery framework using heuristic rules. Hang and Zou [3] produce a collection of complete business processes from the three-tier of e-commerce systems and visualize them using commercial business process modeling tools. Nevertheless, these approaches [1][2][3] require manual verification of the results, and ignore the design structure of UIs when recovering tasks that deliver single unit of work. In this paper we use the dynamic analysis to automatically verify the results from static analysis and enhance the task identification using UI design patterns.

Wil *et al.* [5][6] recover process models from execution logs. Their research effort is used to generate logs. Feature location is a process to identify the parts of source code corresponding to specific functionalities [7]. Orla *et al.* [8] map features to software entities using dynamic analysis. Adrian *et al.* [9] complement dynamic analysis with latent semantic indexing to identify related features in programming language code. In our approach, the detection of UI patterns relies on the static analysis techniques. However, our approach locates tasks using the structures and designs in the UI implementation.

6. Conclusion

In this paper, we propose techniques for recovering business processes from e-commerce applications and for verifying the recovered processes using dynamic analysis. To identify tasks with appropriate granularity and separate different business processes, we use UI design patterns. The patterns abstract the UI structures and recover tasks that can deliver a single unit of functionality. The usage scenarios of a user are recovered using dynamic analysis and are compared with the results from static analysis. The case study on an open source e-commerce system demonstrates the feasibility of our techniques. In the future, we plan to apply the proposed techniques on other open source e-commerce systems.

Reference

[1] H. Huang, W.T. Tsai, S. Bhattacharya, X.P. Chen, Y. Wang, and J. Sun. “Business rule extraction techniques for COBOL programs”, Journal of Software Maintenance 1998, 10(1):3–35.

[2] Y. Zou, T. C. Lau, K. Kontogiannis, T. Tong, and R. McKegney. “Model Driven Business Process Recovery”, In Proceedings of Working Conference on Reverse Engineering, 2004.

[3] M.K. Hang and Y. Zou, “Recovering Workflows from Multi Tiered E-commerce Systems”, Proceedings of International Conference on Program Comprehension, Banff, July 2007. pp.198-207

[4] H. Schmid and G. Rossi, “Modeling and Designing Processes in E-Commerce Applications”, IEEE Internet Computing, January/February 2004.

[5] W.M.P. van der Aalst, T. Weijters, and L. Maruster, “Workflow Mining: Discovering Process Models from Event Logs”, IEEE Transactions on Knowledge and Data Engineering, v.16 n.9, pp.1128–1142, Sep. 2004

[6] A.J.M.M. Weijters and W.M.P. van der Aalst, “Process mining: discovering workflow models from event-based data”, in: Proceedings of the Belgium-Netherlands Conference on Artificial Intelligence, 2001, pp. 283–290.

[7] D. Poshyvanyk, A. Marcus, and V. Rajlich, “Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval”, IEEE Transactions on Software Engineering, v.33, n.6, June 2007.

[8] O. Greevy and S. Ducasse, “Correlating features and code using a compact two-sided trace analysis approach”, Proceedings of CSMR 2005, 2005, pp.314–323.

[9] A. Kuhn, O. Greevy, and T. Gîrba, “Applying semantic analysis to feature execution traces”, Proceedings of PCODA, Nov. 2005, pp. 48–53..

[10] D. Sinnig, “The Complicity of Patterns and Model-Based UI Development”, Mater thesis in Department of Computer Science, University of Concordia, Montoreal, Canada, 2004.

[11] http://ofbiz.apache.org/