

**BUSINESS PROCESS RECOVERY USING UI DESIGN PATTERNS
AND CLONE DETECTION IN BUSINESS PROCESSES**

by

Jin Guo

A thesis submitted to the School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
(October, 2008)

Copyright ©Jin Guo, 2008

Abstract

A business application automates a collection of business processes. A business process describes how a set of logically related tasks are executed, ordered and managed by following business rules to achieve business objectives. An “online book purchase” business process contains several tasks such as buying a book, ordering a book, and sending out promotions. In this ever changing business environment, both of business applications and business processes are modified to accommodate changed business requirements and improve the performance of the organization. These continuous modifications introduce problems in the following two aspects: 1) Business process definitions are rarely updated to reflect the current business processes deployed in business applications. 2) Business processes may be cloned (e.g., copied and slightly modified) to handle special circumstances or promotions. Identifying these clones and removing them help improve the efficiency of an organization. However, business processes are defined with textual languages that cannot be automatically understood.

To maintain business process definitions up to date, we present our techniques that automatically recover business processes from UIs of business applications and identify clones in the recovered business processes. We leverage UI design patterns, which present the best practices of UI designs, to capture business processes from UIs. To refine the recovered business processes and mark the functionally equivalent tasks, we use existing code clone detection tools, such as CCFinder and CloneDR, to detect clones in business applications, and lift clones from code level to business process level. The effectiveness of our techniques is demonstrated through a case study on 15 large open source business applications.

Acknowledgements

I gratefully thank my supervisor Dr. Ying Zou, not only for her supervision for this research work, but also for her patience and help which encourage me to complete my study.

I am very thankful to Mr. Kingchun (Derek) Foo and Ms. Liliane Barbour for their great contributions to the implementation of our research prototype tool: business process explorer. Kingchun (Derek) Foo put lots of efforts to verify the recovered business processes and give me feedback for the improvement of the recovery techniques.

I would like to thank Mr. Maokeng Hung who constructed the foundation of our business process explorer tool. I also would like to thank all the members in the Software Reengineering Research Group: Mr. Xulin Zhao, Mr. Hua Xiao, Mr. Brain Chan, Mr. Lionel Marks, Mr. Kingchun (Derek) Foo, Ms. Liliane Barbour, Mr. Jeff Beiko, and Mr. Michael Lerner, for making my master study colorful.

I would like to express my gratitude to my committee members: Dr. Ahmad Afsahi, Dr. James Cordy and Dr. Juergen Dingel, for their valuable comments and feedback on this thesis.

Finally, I am grateful to my husband, parents, sister and friends for their support and encouragement.

Table of Contents

Abstract	ii
Acknowledgements.....	iii
Table of Contents	iv
List of Figures	vii
List of Tables	ix
List of Abbreviations	x
Chapter 1 Introduction	1
1.1 Background.....	1
1.1.1 Business Applications	1
1.1.2 UI Design Patterns	2
1.1.3 Business Processes.....	3
1.1.4 Task Clones in Business Processes.....	5
1.2 Problem Definitions	7
1.3 Challenges.....	8
1.4 Thesis Overview	11
Chapter 2 Related Work.....	13
2.1 Design Pattern Recovery.....	13
2.2 Business Process Recovery and Business Logic Identification	14
2.3 Design and Architecture Recovery	16
2.4 Reverse Engineering and Reengineering of UIs	17
2.5 Feature Location and Identification	18
2.6 Clone Detection	18
Chapter 3 Framework for Business Process Recovery and Clone Detection	22
3.1 Representation of Business Processes.....	22

3.2 Steps for Recovering Business Processes and Detecting Task Clones	22
3.3 A Schema for Representing Recovered Business Processes.....	24
Chapter 4 Recovering Business Processes from User Interfaces.....	26
4.1 Identifying UI Design Patterns	28
4.2 Identifying Tasks	34
4.3 Identifying Control Flows.....	36
Chapter 5 Clone Detection in Business Processes	38
5.1 Detecting Clones from UI Tiers.....	38
5.2 Detecting Clones from Business Logic Tiers	40
5.3 Measuring Similarity between Tasks.....	41
5.4 Grouping Task Clones	42
Chapter 6 Overview of Business Process Explorer Tool.....	52
Chapter 7 Case Studies	57
7.1 Application Studied	57
7.1.1 Characteristics of SequoiaERP and Opentaps	58
7.1.2 Characteristics of SMaCS	59
7.2 Business Process Recovery.....	60
7.2.1 Measuring the Effectiveness of Business Process Recovery Techniques.....	60
7.2.2 Procedure for Evaluating Business Process Recovery Techniques	61
7.2.3 Analyzing Results for Business Process Recovery.....	61
7.2.3.1 Analysis of Process Recovery Results for Studied Projects	62
7.2.3.2 Analysis of the Use of UI Design Patterns for Identifying Tasks.....	63
7.2.4 Threats to Validity and Limitations	64
7.3 Clone Detection	65
7.3.1 Measuring the Effectiveness of Task Clone Detection Techniques.....	66

7.3.2 Procedure for Evaluating Clone Detection Techniques	66
7.3.3 Analyzing Task Clone Results from UI Tiers.....	67
7.3.3.1 Analysis of Task Clones Located in the UI Tier of SequoiaERP	68
7.3.3.2 Analysis of Task Clones Located in the UI Tier of Opentaps	69
7.3.3.3 Analysis of Task Clones Located in the UI Tier of SMaCS	70
7.3.4 Analyzing Task Clone Results from Business Logic Tiers	71
7.3.4.1 Analysis of Task Clones Located in the Business Logic Tier of SequoiaERP	72
7.3.4.2 Analysis of Task Clones Located in the Business Logic Tier of Opentaps	73
7.3.4.3 Analysis of Task Clones Located in the Business Logic Tier of SMaCS	75
7.3.5 Summary of Task Clone Detection Techniques	76
Chapter 8 Conclusion and Future work	77
8.1 Thesis Contributions	77
8.2 Limitations and Future Work.....	78
Bibliography	81

List of Figures

Figure 1-1. Three-tier architecture of business applications.....	2
Figure 1-2. An example browse pattern.....	3
Figure 1-3. An example business process for purchasing books on-line	3
Figure 1-4. An example of task clones and code clones	6
Figure 3-1. The overall framework for recovering business processes and detecting task clones.	23
Figure 3-2. Schema for representing the recovered business processes	24
Figure 4-1. Overall steps for recovering high-level processes from screens	26
Figure 4-2. Schema used to describe the structure of a screen	27
Figure 4-3. An example of annotated screens of SequoiaERP	27
Figure 4-4. Screenshot of a main menu pattern instance	30
Figure 4-5. Screenshot of a wizard pattern instance	31
Figure 4-6. Screenshot of a multiple-value input form pattern instance	32
Figure 4-7. Example high-level process vs. low-level process	35
Figure 5-1. Steps for detecting task clones in business process definitions.....	38
Figure 5-2. Pretty-print UI code.....	39
Figure 5-3. Rewrite code fragments from a class file	40
Figure 5-4. Steps for extracting sub-processes	44
Figure 5-5. An example of extracting sub-processes	46
Figure 5-6. An example for refining processes	47
Figure 5-7. Algorithm for initializing sub-processes	48
Figure 5-8. Algorithm for extending sub-processes.....	48
Figure 5-9. Algorithm for extending sub-processes by including sequent neighbor nodes	49
Figure 5-10. Algorithm for extending sub-processes by including alternative neighbor nodes	49
Figure 5-11. Algorithm for extending sub-process by including identical control structures	50

Figure 5-12. Algorithm for merging extended sub-processes into list.....	51
Figure 5-13. Algorithm for refining sub-processes.....	51
Figure 6-1. An annotated screenshot of the business process explore	52
Figure 6-2. Steps for detecting task clones from high-level processes	54
Figure 6-3. Steps for detecting task clones from low-level processes	55
Figure 6-4. Visualizing recovered processes in IBM WBM.....	56

List of Tables

Table 4-1. Features of UI design patterns	29
Table 5-1. Preliminary list of structures for grouping tasks.....	43
Table 7-1. Characteristics of the subject applications.....	58
Table 7-2. Evaluating the recovered high-level processes.....	62
Table 7-3. Tasks recovered using UI design patterns	64
Table 7-4. Performance of our approach for detecting task clones from the UI tier using different thresholds.....	67
Table 7-5. Summary of detected task clones from the UI tier	67
Table 7-6. Performance of our approach for detecting task clones from the business logic tier using different thresholds.....	71
Table 7-7. Summary of detected task clones from the business logic tier	72

List of Abbreviations

AST	Abstract Syntax Tree
BPE	Business Process Explorer
BPMN	Business Process Modeling Notation Specification
CRM	Customer Relationship Management
ERP	Enterprise Resource Planning
FTL	Freemarker Template Language
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
JSP	JavaServer Page
OO Source Code	Object Oriented Source Code
SVG	Scalable Vector Graphics
UI	User Interface
UI Design Pattern	User Interface Design Pattern
UML	Unified Modeling Language
IBM WBM	IBM WebSphere Business Modeler
XML	eXtensible Markup Language
XPath	XML Path Language
XPDL	XML Process Definition Language

Chapter 1

Introduction

1.1 Background

Business applications implement business processes, which describe the order of executing tasks, for the daily operations of an organization. Business processes and business applications keep on changing due to market competitions. In the business domain, business processes are often changed to accommodate new business initiatives and improve the performance of the organizations. In the software domain, business applications are modified to add new sophisticated features. However, the business process definitions are rarely updated to reflect current business processes deployed in business applications. This results in inconsistencies between as-documented and as-implemented business processes. Since this discrepancy may cause business analysts with insufficient knowledge of the deployed business processes and business applications, business analysts may copy an established business process or a business task and modify it slightly to experiment with new initiatives (e.g., different business flows). In this way, task clones, which deliver the same or similar business functions, are prone to be introduced. In the following sub-sections, I give an overview of the background.

1.1.1 Business Applications

Business applications enable organizations to automatically perform their daily operations and processes, such as catalog management, order handling, and campaign promotions. Business applications are often implemented using multi-tier architecture which contains a user interface (UI) tier, a business logic tier and a database tier, as shown in Figure 1-1. In the User Interface (UI) tier, a user interacts with a business application by filling and submitting requests using a web browser. The business logic tier coordinates requests, makes logical decisions and performs calculations. In the business logic tier, a controller captures user requests and dispatches them to

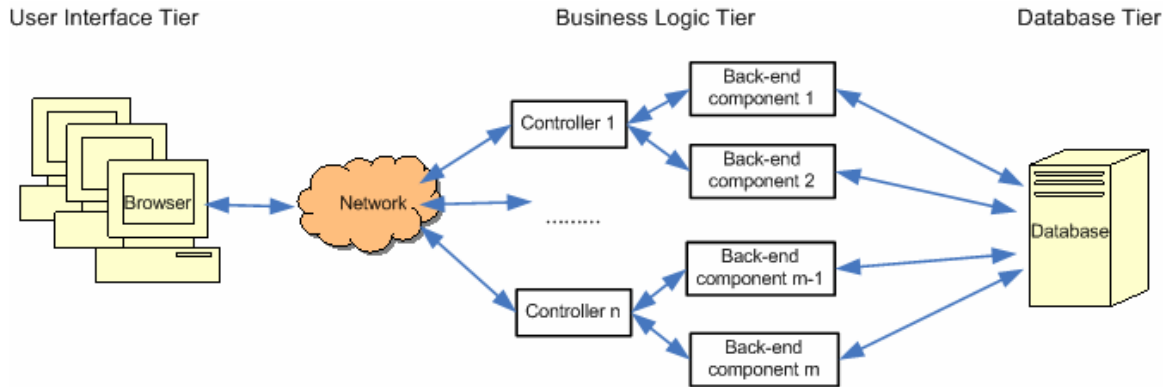


Figure 1-1. Three-tier architecture of business applications

the appropriate back-end components for processing. A back-end component implements one or more business tasks. A controller also determines the flow of execution by either invoking another back-end component to execute tasks or requesting additional user input by loading another UI screen. While processing a user request, the back-end component may communicate with the database tier to retrieve or update data from the database.

1.1.2 UI Design Patterns

In the UI tier, business applications nowadays often provide web-based UIs, which accept input and provide output by generating web pages. The web-based UI includes various widgets (e.g., forms, links and tables) which are interface elements supporting different UI functionalities such as input, output and navigation. UI design patterns, which describe the functionalities delivered through groups of UI widgets [29] and use the best practice to solve specific problems in UI designs, are commonly applied in the UI tier of business applications to better deliver the functionalities. For example, browse patterns [68], which present a set of objects in a tabular format, are generally adopted by online shopping applications to display the available products. As shown in Figure 1-2 from the Nokia online store [57], a list of cell phone plans and the related attributes are displayed in a table to allow users to browse. As useful tools for designing user interfaces, UI design patterns are summarized and collected in [19][68][74][75].

Deal includes	Price plan	Minutes	Texts	Line rental	Effective monthly cost	Deal Cost
Nokia 5310 Red - Vodafone  18 month contract length	Anytime Talk 20  18 month contract length	100 minutes per month	50 texts per month	£20.00 line rental per month	£20.00 is the effective monthly cost for 18 months. Total cost over contract length £360.00	Price today FREE Buy Now
Nokia 5310 Red - Vodafone  18 month contract length	Anytime Text 20  18 month contract length	50 minutes per month	150 texts per month	£20.00 line rental per month	£20.00 is the effective monthly cost for 18 months. Total cost over contract length £360.00	Price today FREE Buy Now
Nokia 5310 Red - Orange  18 month contract length   	Dolphin £25  18 month contract length	200 minutes per month	Unlimited texts per month	£25.00 line rental per month	£20.83 is the effective monthly cost for 18 months. Total cost over contract length £375.00	Price today FREE Buy Now

Figure 1-2. An example browse pattern [57]

1.1.3 Business Processes

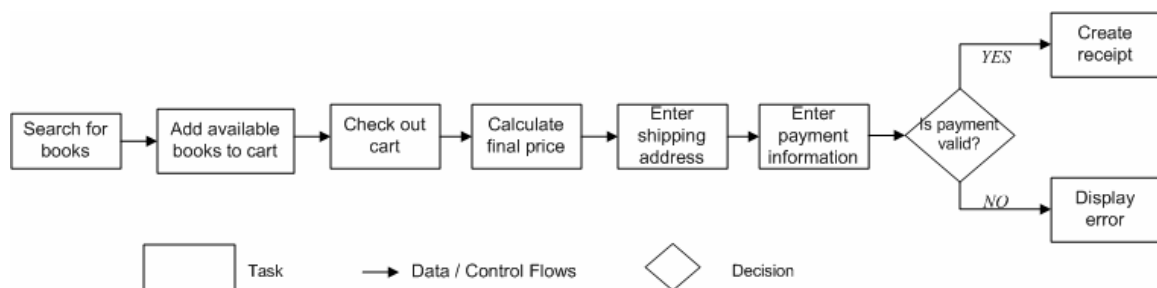


Figure 1-3. An example business process for purchasing books on-line

A business process describes how a set of logically related tasks are executed, ordered and managed by following business rules to achieve business objectives. Business applications implement a set of business processes to fulfill business goals. For instance, an on-line bookstore implements several business processes such as buying a book and managing the inventory. These processes are composed of several tasks. For example, as shown in Figure 1-3, the purchase-book process would contain tasks, such as searching for books, adding the selected books into a shopping cart if the books are available, checking out and validating the buyer's payment approaches (e.g., checking credit limits). These tasks are connected and governed by business rules which are continuously modified and optimized.

Business analysts create, visualize, and analyze business processes using business process modeling tools, such as IBM WebSphere Business Modeler (WBM) [37]. Business process definitions are often described in proprietary formats used by particular business process modeling tools. Business processes can also be specified using standards, such as XPDL (XML Process Definition Language) [81] or BPMN (Business Process Modeling Notation Specification) [10]. A business process definition consists of four major components:

- **Tasks** describe the lowest level of detail needed to achieve a business activity, for example, “Check credit limit”, or “Calculate final price”. A task can be an automatic task that is implemented in the back-end components of business applications and can be automatically executed, or a human task that is delivered by UI screens of business applications and requires human interactions. Users interact with human tasks by providing data. Human tasks in turn invoke automatic tasks which accept inputs, perform functionalities, and return outputs (to other tasks, users or databases). Automatic tasks are implemented in various programming languages, such as Java.
- **Sub-processes** are a group of interrelated simpler tasks. For example, a “Get payment address” sub-process can contain several simpler tasks, such as “Get the receiver’s name”, “Get billing address”, and “Get postal code”. A sub-process can be further divided into more sub-processes and tasks.
- **Data flows** describe the input and output data for a task or a sub-process. For example, tax information and product price can be the inputs for the “Calculate final price” task.
- **Control flows** define routing constraints for executing tasks. We list below examples of routing constraints:
 - 1) *Sequence* – one or more tasks (or sub-processes) and control flow constructs connected in a sequential order. Their sequence of execution is unconditional. For

instance, a “Select payment method” task must be made before we can move to the “Shipping goods” task.

- 2) *Alternative* – describes a single thread of control which selects an execution path among two or more alternatives. For instance, a customer can pick a payment method during the check-out process. A *Decision-relation*, which describes two alternative paths, is a special case of the alternative relation.
- 3) *Loop* – is used for repeating the execution of one or more tasks. The loop constraint has a termination condition. For instance, a customer can keep on putting items in a shopping cart until the customer decides to check-out.
- 4) *Parallel* – allows two or more threads of control to proceed autonomously and independently until all threads of control are merged once they are completed. For instance, both the customer and the inventory department must be informed when an ordered item is out of stock.
- 5) *Pre- and post- conditions* represent entry and exit conditions for a particular sequence of tasks.

1.1.4 Task Clones in Business Processes

Business processes are often designed for a particular department, an organization or a domain. The knowledge is seldom shared across the boundary of the departments, organizations, and domains. To optimize business processes, a business analyst needs to manually identify repeated business tasks across multiple processes in order to reduce redundant processing steps and improve the overall performance of the organization. This requires business analysts to recognize task clones. More specifically, task clones refer to tasks which deliver the same or similar business functions across multiple processes. Business processes are usually cloned as well to mitigate the risk associated with modifying complex and critical business processes [16]

or when experimenting with new initiatives in business processes. These cloned processes are rarely integrated back into the original processes leading to duplicate business processes. By analyzing the code corresponding to a task, we can detect task clones, since task clones have similar implementation involving code clones, which are identical or almost identical code fragments. Looking at Figure 1-4(c) and Figure 1-4 (d), we identify that *View shipment info* and *Browse shipment info* tasks are implemented by two cloned HTML tables shown in Figure 1-4(a) and Figure 1-4(b). Similarly, the two instances of the *Create order invoice* task are implemented by code clones presented in Figure 1-4(e) and Figure 1-4(f).

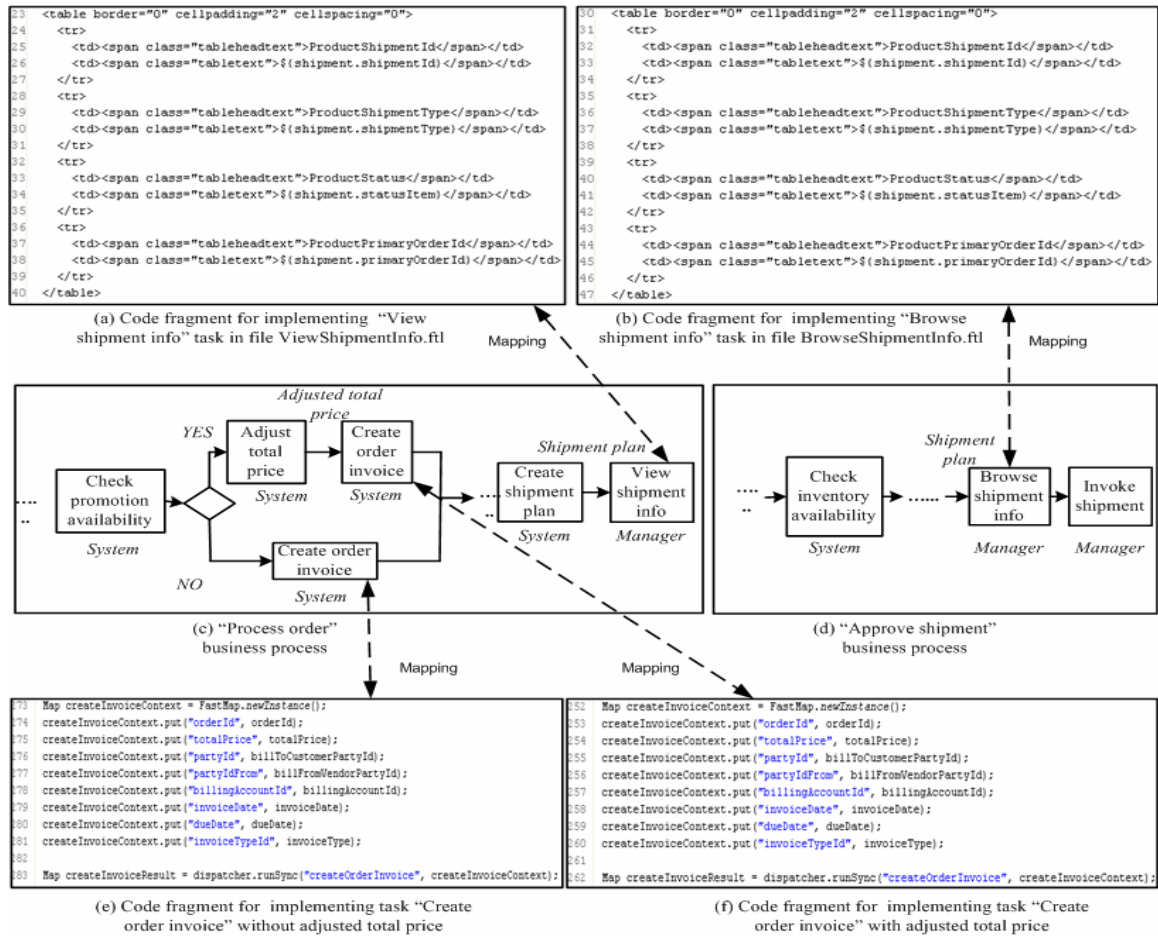


Figure 1-4. An example of task clones and code clones

1.2 Problem Definitions

We identify the problems associated with business process definitions and business applications.

1) *Inconsistencies between As-documented and As-implemented Business Processes*: this discrepancy introduces difficulties for business analysts to modify or optimize business processes and for software developers to implement the changes proposed by business analysts. Without a correct understanding of current deployed business processes (i.e., as-implemented business processes), business analysts cannot effectively propose changes to improve the performance of business applications or accommodate new business requirements. Without an up-to-date business process definition, which explicitly denotes links between the tasks specified in process definitions and the corresponding implementations in the business applications, it is difficult for software developers to identify the appropriate code segments that must be modified in response to changed business processes.

2) *Task clones in business processes*: as a business application grows in size and complexity, these task clones increase the cost of maintenance and require refactoring at the source code and business process levels. If software developers implement the required changes or fix bugs in the implementation of a task, it may be necessary to locate and update the implementation of other task clones by replicating the changes. Due to the complexity of business processes and the size of business applications, automatic techniques are needed to identify such clones at the business process level. For business analysts, the cloned tasks with different names make the processes difficult to understand. Changes in task clones require software developers to manually locate all the code fragments that contribute to the task clones. To reduce the costs and improve the productivity of business process optimization and software maintenance efforts, it is beneficial to detect task clones and refine business processes to explicitly denote and document task clones.

1.3 Challenges

To recover complete business processes, Hung and Zou [35][36] propose a business process recovery approach that extracts business processes from the multi-tiered architecture of business applications, including the UI tier and business logic tier. When recovering a business process from UIs, Hung and Zou [35][36] start from an initial UI page, map UI widgets (e.g., forms) into tasks, and identify control flows through tracing page navigation flows. However, the techniques proposed by Hung and Zou [35][36] to recover business processes from UIs have a few limitations. First, the process recovery techniques [35][36] simply map a widget into a task according to heuristic rules and this may lead the recovered business tasks to be too trivial, since multiple widgets can often be used together to deliver one functionality. Second, the process recovery techniques [35][36] trace page navigation flows to identify control flows among tasks delivered in multiple pages, but a lot of page navigations are designed to allow users to quickly navigate and locate resources, and therefore cannot reflect the execution sequence among tasks. For example, through clicking menu items on top of a UI page, users can switch from one component to another, but there is no sequence control flow among tasks delivered by pages in these components. Finally, without automatic approaches to identify initial pages for business processes, the recovery techniques [35][36] start to extract business processes from UI pages specified by humans. This requires tremendous human work to review and analyze UI pages.

Little effort is spent to identify the duplicated entities (e.g., duplicated tasks and duplicated processes) in the recovered results. These duplicated entities share similar implementations, deliver similar functionalities, and require extra maintenance efforts. More specifically, task clones resulted from similar code implementations may commonly exist in the business processes recovered from the UI and business logic tiers. These task clones often require extra maintenance and restructuring processes, and therefore, need to be identified and explicitly denoted.

My thesis extends the prior work [35][36], provides effective techniques to identify tasks from the UI tier, and refines recovered business processes to identify task clones. We focus on addressing the following challenges:

- 1) ***The arbitrary UI page transitions make the identification of control flows from UIs difficult.*** In order to ease page navigation, UI pages often provide hyperlinks that connect one page with another. Therefore, it is difficult to identify the control flow constructs (e.g., sequence, alternative and parallel) between the functionality delivered in different pages in a business application. It is also hard to determine the starting point of a business process from the linked pages. In this thesis, we leverage UI design patterns to recover control flows from UI page transitions and identify the starting page of a business process. We use navigational patterns, which describe the navigational structure of screens, to identify the sequence control flows between tasks delivered in sequential pages. We use structural patterns, which organize the overall structure of UI screens, to identify the starting page of a business process.
- 2) ***The hierarchical structure of a widget complicates the identification of a task from UIs with appropriate granularity.*** Many possible tasks with different levels of granularity can be identified from the UI widgets. For example, a table widget contains multiple cells each of which can further encapsulate several widgets (e.g., hyperlinks and buttons). 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. Since UI design patterns characterize the functionality delivered through a group of widgets, we use UI design patterns to group a set of widgets into a task with proper granularity.

- 3) ***Comparing task names is not sufficient to recognize task clones, and therefore it is impossible to identify task clones by reviewing process definitions.*** The details of the functionality of a task are omitted. In [54], the function name is used as a comparison point to detect function clones. However, names for our recovered tasks are informally specified using simple verbs and nouns, and cannot be used as criteria to detect task clones. Two functionally identical tasks (e.g., *Find a book* and *Search books*) could be named very differently. In other cases, two tasks with the same specified name in process definitions may have different implementations delivering distinct functionalities. Such tasks are not task clones. To identify task clones, we start to detect clones in source code and lift clones to task level through the traceability between business tasks and their implementations in business applications.
- 4) ***Business applications implement a large number of business processes, and therefore a manual process for identifying task clones is not feasible.*** In order to identify task clones, software developers need to manually locate and compare the implementation of tasks. This is tremendous work due to the large amount of business tasks and the complexity of business applications. To make the task clone detection efficient, we adopt existing code clone detection tools (i.e., CCFinder [41] and CloneDR [15]) to locate code clones in business applications and automatically parse the code clone results to identify task clones.
- 5) ***Business applications are multi-tiered applications written in a myriad of languages.*** The implementation of each business process is scattered throughout various tiers. We must detect similarities across the different tiers. Human tasks performed using the UI screens can be implemented using HTML, JSP and XML. Automatic tasks conducted by the business logic tier are written in Java or other high level programming languages. It is challenging to detect code clones using a single clone detection tool due to the various programming languages used for implementing tasks. According to features of

programming languages applied on different tiers of business applications, we leverage different code clone detection tools to identify code clones.

- 6) *Clone detection techniques operate at the code level however business analysts are not familiar with the code.* Moreover, it is tremendous work for business analysts to analyze the usage patterns of task clones to identify clones in process level (e.g., sequences of task clones). Business analysts are more familiar with the business process definitions. Therefore we must present the results of our analysis at the process definition level. Moreover, the clones identified from the code may contain implementation details such as utility functions or API calls. Business processes convey high-level business relevant functions. The low-level implementation detail needs to be filtered. Clones in process level are usually more valuable than clones in task level when business analysts are trying to optimize large segments of business processes and improve the performance. We first represent the clones in business task level with which business analysts are familiar. To ease the optimization of business processes, we recognize the usage pattern of task clones to group task clones into sub-processes which coordinate task clones and identical tasks with the same control flows.

We use UI design patterns, which describe the best practice for UI designs, to capture tasks and control flows delivered by UIs. We analyze screens and navigation flows to identify business tasks and their dependencies on data and controls. With the help of UI design patterns, we also identify the initial pages of a business process starting from UIs. We apply the business processes recovery techniques proposed in [35][36] on the business logic tier and database tier. As a result, we can recover complete business processes from a business application.

1.4 Thesis Overview

The remaining chapters of this thesis are organized as follows:

CHAPTER 1. INTRODUCTION

- Chapter 2: We review related work about design pattern recovery, design and architecture recovery, reverse engineering of UIs, feature locations and clone detection.
- Chapter 3: We describe the framework of our business process recovery and task clone detection approach. To give readers an overview about the steps to recover complete business processes, the framework generally introduces the business process recovery steps applied on both of user interfaces and back-end components of business applications. We further introduce the representation and definition of the recovered business processes
- Chapter 4: We depict the strategies for recovering business processes from UIs. We also detail the recovery steps including recognizing UI design patterns, extracting tasks, and identifying control flows.
- Chapter 5: We describe the strategies to identify task clones in business processes recovered from UIs and back-end components.
- Chapter 6: We give an overview of our prototype tool: Business Process Explorer (BPE).
- Chapter 7: We present our case studies and evaluate our experimental results.
- Chapter 8: We conclude our work and discuss future work.

Chapter 2

Related Work

2.1 Design Pattern Recovery

Design patterns are solutions of common non-trivial design problems and can be reused frequently [47]. In Object Oriented (OO) source code, design patterns can be described by a set of classes and the relations among these classes [1]. The well known design patterns applied in OO source code are presented by Gamma *et al.* in [28]. Design pattern recovery can ease the understanding of program and designs [47][42][1]. Antoniol *et al.* [1] propose a multi-stage reduction strategy based on software metrics and structural properties to recover design patterns from Object Oriented (OO) source code. Krämer and Prechelt [47] develop a system named Pat to recover design pattern instances through searching the Prolog representation of design pattern repository and software design information. Keller *et al.* [42] represent the source code with a UML-based format, which is parsed to get the repository of source code information, and implement query mechanisms to recognize design patterns from source code information. Costagliola *et al.* [18] extract the class diagrams and represent them with Scalable Vector Graphics (SVG) from OO source code and use visual language parsing approach to recover design patterns.

All above design pattern recovery approaches are applied on OO source code through matching software design information and design patterns mentioned in [28]. Different from above pattern recovery work, we recovery UI design patterns from UI source in order to capture the functionalities delivered by UI widgets. In our UI design pattern recovery approach, we summarize the features of UI design patterns, and analyze the UI code to identify the structure of the UI which correspond to a particular UI design pattern. UI design patterns are commonly applied in the GUI (graphical user interface) designs, which offer graphical icons and visual

indicators to fully represent the information and actions available to a user. The correct use of UI design patterns eases the design and development processes [68]. UI design patterns are summarized and analyzed in both industry [82] and academic areas [68][75]. UI design patterns are published in websites [19][82][75], books [74], related workshop [78] or other research works[68].

2.2 Business Process Recovery and Business Logic Identification

Business processes or business rules recovery techniques are useful for business application maintenance, evolution and migration, especially when the software documents have been out of date. Huang *et al.* [33] propose business rule extraction approaches using variable classification, program slicing and many other software maintenance techniques. Sneed and Erdos [70] extract business rules in source code by using data results as entry points and tracing assignment statements as well as logic decision. Earls *et al.* [21] implement a tool used for manually extracting business rules starting from error processing sections in source code. Shao and Pound [67] summarize the existing techniques, such as data understanding and program understanding, used for extracting business rules. Zou *et al.* [83] describe a model-driven business process recovery framework using heuristic rules to automatically capture the business process. When identifying business rules or business processes from source code of applications, these approaches [21][33][67][70][83] ignore user interfaces of applications and only focus on back-end components. However, the user interface in business applications nowadays plays an important role to deliver tasks and control flows to users. For example, a user provides data to the UI tier to complete a human task which transfers the request and data to business logic tier; several automatic tasks that are implemented in the back-end components may be invoked to process the data and request, and return the result to the UI tier; the UI tier presents the returned results and the next task to users. Therefore, the user interface in business applications is an indispensable component for delivering complete business processes. Without analyzing the user

interface of business applications, the recovered business processes will only contain tasks and control flows implemented in the back-end components. A lot of tasks and control flows delivered by user interfaces will be omitted and the recovered business processes cannot reflect the complete behavior of business applications. The business process recovery techniques proposed in this thesis are applied on UIs, but previous researches [21][33][67][70][83] recover business processes or business rules from back-end components rather than UIs. This makes it difficult to compare our process recovery results with the business processes or business rules recovered by techniques in [21][33][67][70][83].

To recover complete business processes, Hung and Zou [35][36] analyze multiple tiers of business applications. In the UI tier of business application, Hung and Zou [35][36] map UI widgets into tasks and derive control flows among tasks (or sub-processes) from the order of forms in a page and the sequence of page transitions. To improve the techniques in [35][36] to identify tasks, we leverage UI design patterns to group widgets into tasks to avoid identifying trivial tasks. Similar as [35][36], when we identify control flows among tasks, we leverage the order of patterns within a screen, but we improve the control flow identification approach in [35][36] by considering data dependencies and using UI design patterns. We recover sequent control flows from the page transitions of navigational patterns that describe the navigational structure of UI screens. When we trace page navigations to identify the subsequent control flows, we ignore page transitions invoked by triggers grouped into structural patterns, which organize the overall structure of UI screens. Since these triggers are designed to ease the navigation among different components or business processes, page transitions invoked by these triggers cannot reflect the order among tasks delivered by these pages. The process recovery techniques in [35][36] start to recover business processes from UI pages manually identified by humans. In this thesis, we automatically locate the pages targeted by triggers of structural patterns as the starting point of a business process. After we recover business processes, we refine the process definitions

by detecting and explicitly denoting duplicated tasks, so as to alleviate the workload in software maintenance.

Weijters and Aalst [76] use dynamic analysis and process mining in workflow log events to synchronize the predefined workflow processes and the actual executed workflow processes. Workflows are computer representations of business processes. Different from [76], based on static analysis, we can recover the current deployed business processes even when some components of business applications cannot be executed. Moreover, our process recover approach can still work when there are no available predefined business processes.

2.3 Design and Architecture Recovery

Design recovery leverages source code, domain knowledge, documentation and personal experience together to create an abstraction of subject applications [7][14]. Biggerstaff [7] describes a semiautomatic model-based design recovery system. Sneed [72] describes a project which bridges the gap between programs and specifications. This project leverages static analysis and tools to re-document and re-specify programs, and conducts testing against the new specifications. Sneed and Jandrasics [71] describe an approach to translate source code to specifications expressed using an intermediate design schema and entity/relationship model.

Software architecture recovery focuses on the architectural information and supports system understanding [24]. Eixelsberger *et al.* [24] present a framework to recover software architecture which is viewed as a set of architecture properties (e.g., parameter passing and variables). The recovery methods involve existing tools and domain knowledge. Hassan and Holt [31] present a tool to recover the architecture of web applications and show the relations, including hyperlinks, control and data dependencies, between various components.

The above design and architecture recovery work provides a high-level view of applications to help software developers understand or maintain the applications. Different from above work, our business process recovery approach aims at extracting the current business process deployed

in the business application to reflect the behaviours of organizations. The recovery business process can ease the work of both of software developers and business analysts.

2.4 Reverse Engineering and Reengineering of UIs

Reverse engineering is applied on UIs so as to extract useful information for testing or understanding the interface. Memon *et al.* [55] apply reverse engineering on a GUI to get the navigation flows among windows represented as a GUI forest and extract the event-flow graphs to describe the relation among UI events. Ricca and Tonella [62] create the analysis model of a web site from the navigation and interaction patterns. Hilbert and Redmiles [32] summarize the existing approaches, such as sequence detection and sequence characterization, utilized to extract usability information from UI events.

UIs can be restructured or reengineered in order to migrate from one platform to another or improve user interfaces. Moore *et al.* [56] describe steps to migrate the text-based UIs of legacy applications to GUIs using domain model and pattern matching. In [73], the CelLEST project uses a suite of methods, based on understanding and modeling uses interactions with the legacy-application interface to migrate the text-based UIs of legacy applications to GUIs. Dynamic analysis is used in [56][73] in order to extract the state transitions or UI events. Zou *et al.* [84] reengineer user interfaces based on business processes to improve the usability of e-commerce applications. Heuristic rules are applied to recover the binding between UI components and tasks specified in business processes. Then, a dynamic execution environment is generated in order to guide users through a sequence of UI components fulfilling tasks sequences.

Different from above research, our reverse engineering on UIs focuses on using static analysis to recover UI design patterns. In the UI design pattern recovery approach, we recover the navigation flows among UI pages, trace the data flow among different UI widgets, and identify different UI design patterns based on their own features.

2.5 Feature Location and Identification

Feature locations are localizing source code parts corresponding to specific functionalities [60]. Chen and Rajlich [13] propose a feature location approach which searches the program dependency graph and requires human assistance. Poshyvanyk *et al.* [60] leverage latent semantic indexing and scenario-based probabilistic ranking to search and locate features in the source code. In both of these researches, human work is required to make decisions or create queries. Our process refinement research develops automatic techniques to identify similar business functionality features.

Salah *et al.* [65] use execution profiling to locate features in the Mozilla web browser. Eisenbarth *et al.* [23] adopt combined hybrid techniques such as execution profiling and domain analysis to locate features. Involving dynamic analysis, this approach requires that programs can be compiled and exercised under different scenarios. Based on static analysis, our task clone detection approach can still find the similar functionalities features which cannot be executed correctly.

Some researches use feature comparison. Salah *et al.* [65] find similar features by checking the method invocation relationship in scenario execution. Antoniol and Guéhéneuc [2] compare features using model transformation techniques. We compare the similarity of business tasks based on the source code similarity.

2.6 Clone Detection

Code clones are identical or almost identical code fragments. Code clones commonly exist in large software applications. Baker [3] and Laguë *et al.* [48] respectively report on 6%--13% of code clones in large computer software based on the number of functions or the number of lines of code ignoring comments and white spaces. Code clones are generally considered to be harmful in software maintenance [3][5][6][41][44], due to the fact that bug fixing and code modifications are usually required to be replicated in multiple versions of code clones [6][41][44]. Code clones

are caused by copy-and-paste actions. In software development and maintenance, programmers may copy the existing code and only modify the copies to accommodate the new purpose, especially when programmers lack full understanding of the program and struggle to avoid introducing problems into the working software. Code clones are detected using various techniques, such as string matching, token comparison, abstract syntax tree comparison, and pattern matching.

String matching techniques are adopted in [3][20][7][40]. Replacing parameter names [3] and identifiers [40] are used to detect almost identical clones. Duploc [20] is a clone detection tool based on string matching. It slightly transforms source code by deleting white spaces and comments and compares code lines to identify clones. Based on string matching, Duploc is not language dependent and can be easily adapted to various languages.

Token comparison is a commonly used technique to detect code clones. CCFinder [41] is a clone detection tool which applies transformation rules to source files written in various languages (e.g., Java, C, C++, COBOL) and detects clones based on token-comparison. In this way, CCFinder can detect identical clones and almost identical clones, such as cloned code portions with different variable names. For source files written in languages (e.g., HTML) to which transformation rules cannot be applied, CCFinder provides a 'PlainText' mode to detect code clones, but only identical clones can be found. Clones is a clone detector provided by the Bauhaus project [9]. Based on token comparison, Clones supports clone detection from applications implemented with C, C++, Java, Ada and COBOL. Both identical clones and almost identical clones, such as copies with renamed identifiers, can be detected. Different from CCFinder, Clones does not provide the 'PlainText' mode to detect clones from source files written in other languages. Since syntax is not considered by token comparison techniques, the detected clones may cross different syntactic units. For example, a code clone may have overlaps with two method declarations.

Abstract Syntax Trees based clone detection has higher precision than clone detections based on string matching or token comparison [6]. CloneDR[15] detects clones based on AST (Abstract Syntax Tree) and it is able to detect both identical clones and almost identical clones. CloneDR can support clone detections in languages such as C, C++, Java, and COBOL.

Using dependence graphs and program slicing, Komondoor and Horwitz [44] detect clones from non-adjacent texts in the program. Software metrics or code metrics are used to detect clones [46][48][54].

Basit and Jarzabek [4] use data mining approach to recognize the pattern of cloned code fragments to infer the similarity in higher levels (e.g., class files). Our clone detection work lifts the clones from code level to higher level (e.g., tasks and process levels), but we use other approaches rather than data mining. We lift the clones from code level to task level through the traceability between source code and business tasks. We continue to lift the clones from task level to process level by recognizing the usage patterns of groups of task clones across multiple processes.

Web applications are prone to be threatened by code clones due to the uninstructed development and maintenance processes as well as the inherent complexity [8][51]. Levenshtein distance is a common technique used in [51][52][63] to detect cloned web pages. In [52] a web page is considered to be a set of predefined features (e.g., sequence of tags or ASP features). The work in [53] applies similarity comparison using Levenshtein distance in content and scripting code levels. Lanubile *et al.* [11][49] detect function clones in scripting code. Cordy *et al.* [17] apply standard lexical comparison tools on pretty-printed HTML code to detect almost identical clones. Roy and Cordy [64] extend the work in [17] to detect near identical clones from C language by introducing flexible pretty-printing, code normalization, code filtering and output generations. We apply similar pretty-printing approach used in [17] on UI code blocks, but we adopt a different clone detection tool (i.e., CCFinder) on the pretty-printed code. Rajapakse and Jarzabek [61] leverage CCFinder [41] to detect clones from any text files included by web

CHAPTER 2. RELATED WORK

applications to get the percentage of clone tokens. Our clone detection approach also applies CCFinder [41] to detect clones. Different from [61], we detect clones from the business relevant code, rather than every text file.

Chapter 3

Framework for Business Process Recovery and Clone Detection

3.1 Representation of Business Processes

A business application typically implements a large number of tasks. To reduce the complexity of understanding the recovered business processes, we represent the business processes in terms of two abstraction levels: high-level and low-level business processes [35][36]:

- 1) A high-level business process gives an overview of a business process recovered from the UI screens and interactions between the UI and business logic tiers. A high-level business process includes human tasks, which require the interaction with users, or sub-processes, which detail the major steps for a complex task implemented by a back-end component or a UI screen. For example, in an online checkout business process, “Find product” and “Enter payment information” are human tasks, while “Create order” is a sub-process that is implemented as a Java method in a back-end component.
- 2) A low-level business process corresponds to a sub-process in a high-level process. The sub-process has a set of automatic tasks performed by back-end components. For example a low-level process for “Create order” would show the need to create an order for each selected item and the need to update the inventory. However, the low-level process would not show any tasks related to converting the user input or error checking (i.e., utility or data conversion steps) since they are not business relevant tasks.

Separating complex processes into two levels provides a clearer view of the overall structure of a business application while hiding processing details.

3.2 Steps for Recovering Business Processes and Detecting Task Clones

It is challenging to recover the complete definition of a business process from its implementation due to the intricate control and data dependencies among the intermixed human

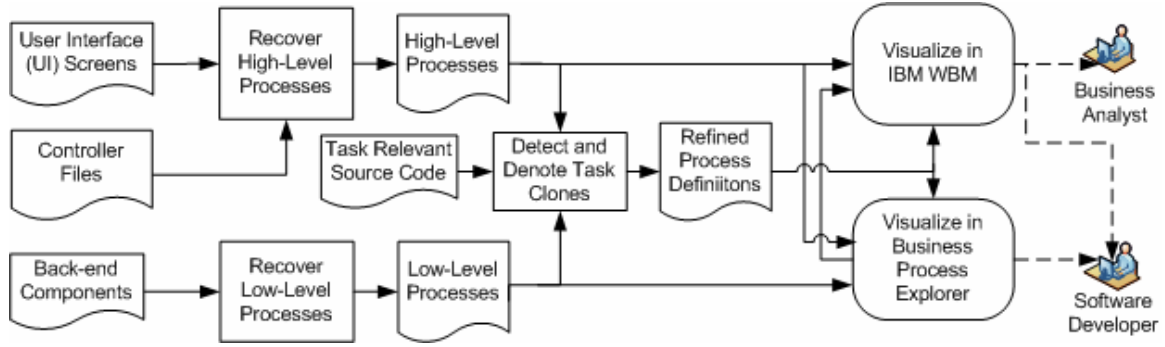


Figure 3-1. The overall framework for recovering business processes and detecting task clones

and automatic tasks. Figure 3-1 gives an overview of our approach for recovering the business process definitions and detecting task clones in the recovered process definitions.

To identify possible execution flows among the UI tier and the business logic tier of business applications, we analyze the execution paths of controller scripts to understand the structure of a business application. By analyzing the controller files and detecting UI design patterns, we locate the starting point for each business process and retrieve the screens and the back-end components that are invoked from these screens. We analyze the screens and controller files to recover high-level processes. To recover low-level processes, we parse the source code of the back-end components in the business logic tier to extract the automatic tasks and their dependencies [35][36]. Database accesses are captured in the business logic tier.

After the business processes are recovered, we apply task clone detection strategies to recognize the similar tasks in the recovered business processes. We leverage existing clone detectors (i.e., CCFinder and ClondDR) to detect code clones from the task relevant source code, which can be mapped into tasks, and lift clones from the code level to the business process level. After task clones are identified, we group task clones and refine the corresponding business process definitions.

3.3 A Schema for Representing Recovered Business Processes

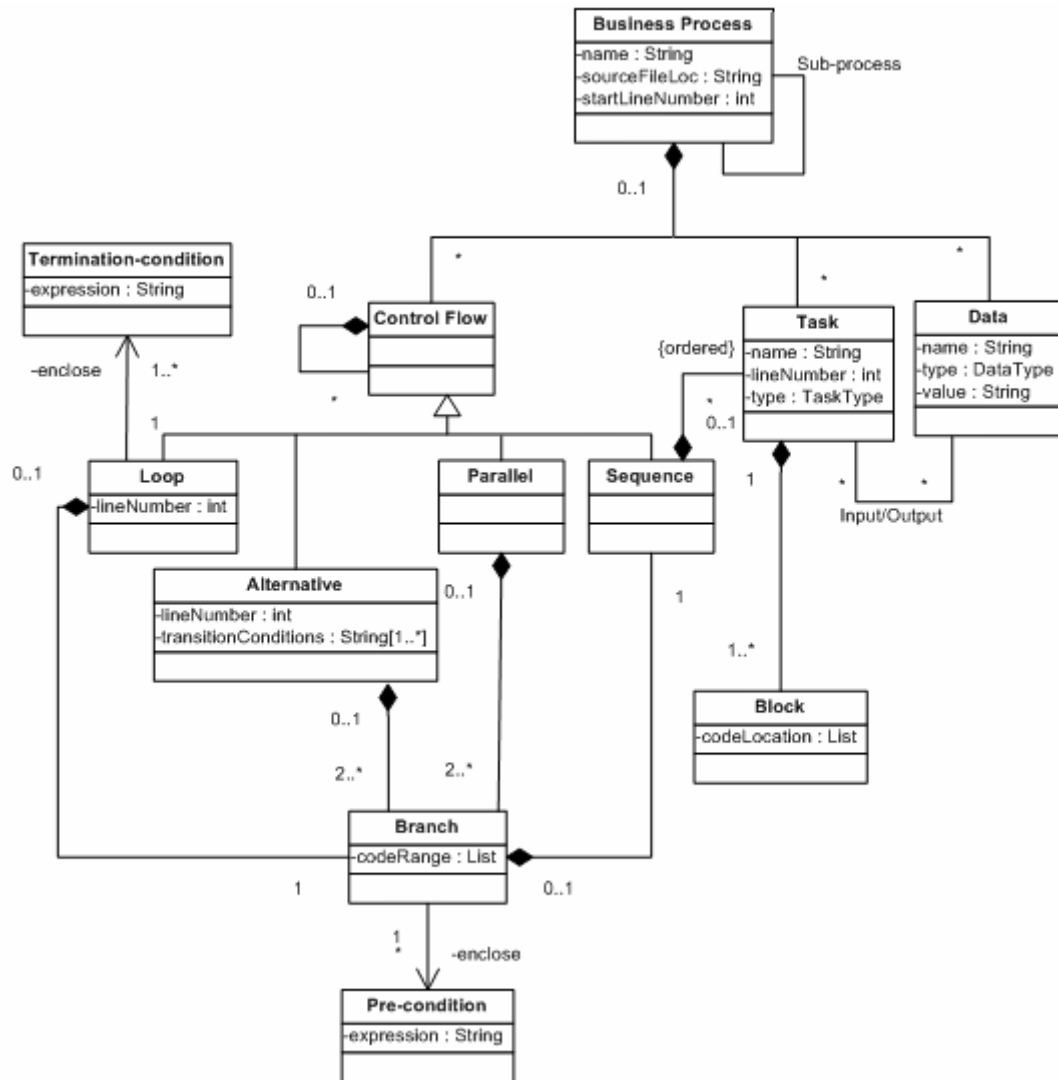


Figure 3-2. Schema for representing the recovered business processes

We define a schema as shown in Figure 3-2 for representing the recovered processes. We keep the relevant software entities (e.g., the starting line of code and the ending line of the code) which correspond to each process entity in a recovered process. For example, for a process entity, we record the source file and the line numbers in the file. Each task includes a block entity, denoting the code portions that contribute to the task. Line numbers are used to record the location of code blocks in business logic tiers mapped into tasks. To create complete traceability

CHAPTER 3. FRAMEWORK FOR BUSINESS PROCESS RECOVERY AND CLONE DETECTION

between tasks and related code blocks in the UI and business logic tiers, we use different schema (e.g., XPath expression [80] for XML element, line number and column number) to record the location of code blocks. Control flow constructs, such as loops and alternatives, contain one or more branches with pre-conditions for the execution of each branch. A branch encloses a sequence of tasks. If the pre-conditions for a loop branch are evaluated to be true, a sequence of tasks in the branch is executed. The termination-conditions are evaluated to decide whether to stop the repeated execution of branches within a loop control construct. A branch contains the code ranges corresponding to the starting and ending lines of the branch in the implementation. The branches in parallel control constructs have no pre- and termination-conditions for their execution order, since they share multiple threads of controls.

Chapter 4

Recovering Business Processes from User Interfaces

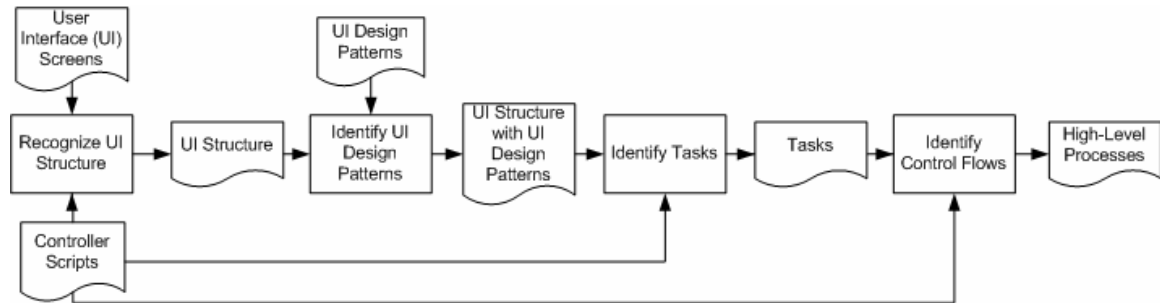


Figure 4-1. Overall steps for recovering high-level processes from screens

Figure 4-1 illustrates the overall steps for recovering high-level processes from the screens and controller scripts. The screens can be written using HTML, XML, JSP, or FTL (Freemarker Template Language) [27]. We developed a parser for each language to recognize the UI structures. A UI design pattern describes a functionality delivered through a group of UI widgets [29]. UI design patterns are re-used in interactive UI designs [56][74][77]. Each screen implements several UI design patterns in the business applications we analyzed. To deal with the complexity of UI designs (e.g., hierarchical widget structures and arbitrary page transitions), we recognize UI design patterns to group UI widgets into more abstract tasks. We also capture the control flows between tasks. As a result, we produce high-level processes.

In the following sections, we detail our approach. Section 4.1 discusses the structure of a screen and the features used for recognizing UI design patterns. Section 4.2 presents our approach for recovering tasks from the UI code. Section 4.3 explains our approach for identifying control flows.

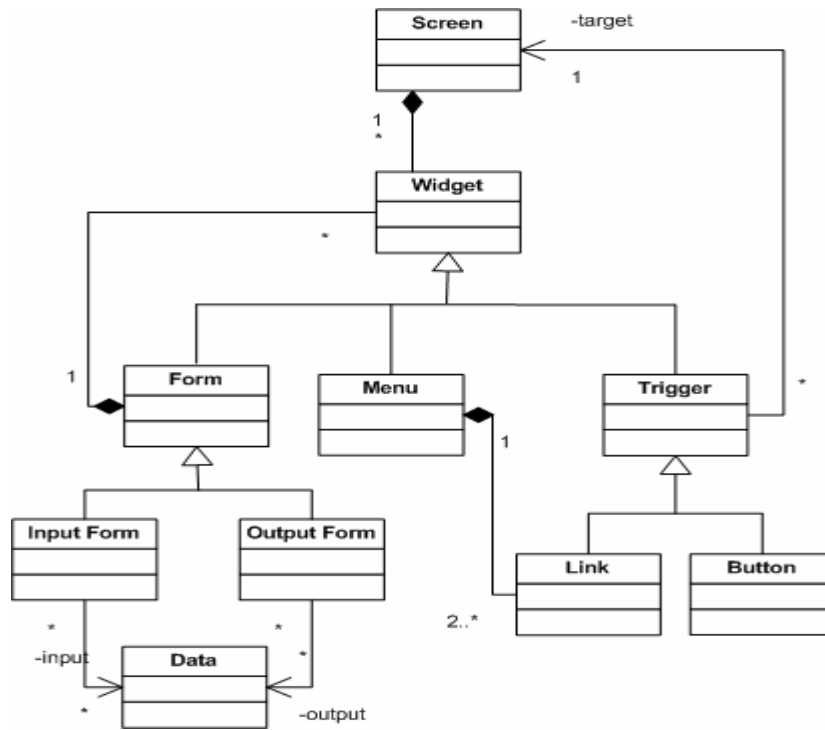


Figure 4-2. Schema used to describe the structure of a screen

The screenshot shows the 'Content Manager Application' interface. At the top, there is a navigation bar with tabs: Accounting, Catalog, Content, Example, Facility, Manufacturing, Marketing, Order, and Partner. Below this is a 'Content Manager Application' header with a 'Menu' button. The main navigation bar includes links: Main, Web Sites, Surveys, Content, DataResource, Content Setup, Data Resource Setup, and Content Setup. A red box highlights the 'Content Manager Application' header and the main navigation bar. A red arrow points to the 'Go to Data Resource' link. The form area contains the following fields:

- Content ID: ARTICLE_WRAP
- Content Type Id: Document (dropdown)
- Owner Content Id: (text input)
- Data Resource Id: ARTICLE_WRAP (text input)
- Template Data: (text input)
- Resource Id: (text input)
- Data Source Id: (text input)
- Status Id: (dropdown)
- Mime Type Id: (dropdown)
- Character Set Id: (dropdown)
- Child Leaf Count: (text input)
- Child Branch Count: (text input)

At the bottom of the form, there are fields for 'Created by User Login' (admin), 'Last Modified by User Login' (admin), and 'Created Date' (2003-12-11). A red box highlights the 'Created by User Login' and 'Last Modified by User Login' fields. A red arrow points to the 'Update' button. The 'Update' button is located at the bottom right of the form. The 'Create New' button is located at the bottom left of the form. A red arrow points to the 'Create New' button. The 'Link' button is located at the bottom right of the form. A red arrow points to the 'Link' button.

(a) Edit Content Screen

Accounting Catalog Content Example Facility Manufacturing Marketing Order Parts

Menu

Content Manager Application

Main Websites Surveys Content DataResource Content Setup DataResource

Find **Content** Association Role Purpose Attribute Metadata

Content Id Equals ☐ Begins With ☐ Contains ☐ Is Empty ☐

Content Type Id

Owner Content Id Equals ☐ Begins With ☐ Contains ☐ Is Empty ☐

Data Resource Id

Template Data Resource Id Equals ☐ Begins With ☐ Contains ☐ Is Empty ☐

Data Source Id Equals ☐ Begins With ☐ Contains ☐ Is Empty ☐

Edit	Content Type Id	Owner Content Id	Data Resource Id	Template Data Resource Id
ARTICLE_WRAP	DOCUMENT		ARTICLE_WRAP	
ASK	WEB_SITE_PUB_PT	WebStoreFORUM		
BLOG_MASTER	DOCUMENT			

Link **Search Pattern**

(b) Find Content Screen

Figure 4-3. An example of annotated screens of SequoiaERP [66]

4.1 Identifying UI Design Patterns

To automate the analysis of screens, we developed a schema for describing the structures of a screen. The schema is shown in Figure 4-2. Each screen is composed of various UI widgets, such as forms, menus and triggers. A form is specialized into two types: input forms and output forms. An input form allows a user to provide information required by back-end components. An output form displays the results from a back-end component. Buttons and links are triggers. A link connects to a new screen, and a button invokes a back-end component which may result in page transitions. For example, in the screen shown in Figure 4-3(a) [66], an “Update” button and three links (e.g., “Go to Data Resource”) are presented. Essentially, forms and menus are high-level widgets that can encapsulate other widgets. Triggers are the widgets at the lowest level of granularity without nested structures.

The complexity of the UI structure causes difficulties in identifying the tasks and the control flows (e.g., sequential and alternative orders) among these tasks in the consecutive screens. The complexity of the UI stems from the complex links and hierarchical structure of UI widgets. Looking at Figure 4-3(a), an input form contains multiple rows. Each row encapsulates widgets: text fields, links, buttons and selection lists. Many possible tasks with different granularities can be identified from the widgets. For example, the entire input form could be considered as a task for editing the information of a specific content. Alternatively, each text field can be considered as a task for editing one attribute of the specific content. Similarly, the “Update” button could be considered as a task that submits the new content to the database. However, a business process is intended to capture business operations with coarser granularity, rather than the detailed implementation steps. A recovered process definition with an excessive number of fine grained and detailed non-business relevant tasks would be difficult to understand and would be of limited value to practitioners. UI design patterns are used to capture a single business task delivered by a group of UI widgets.

We classify UI design patterns into three categories: structural patterns, navigational patterns, and behavioral patterns. In Table 4-1 we summarize the features used to detect each category of UI design patterns. We list example UI design patterns for each category in Table 4-1. We analyze the UI code to identify UI design patterns applied in each screen. We describe our approach below.

Table 4-1. Features of UI design patterns

Category	Pattern instances	Description	Features for Example Patterns
Structural patterns	main menu, double tab navigation, fly-out menu, icon menu[19][74][75]	A set of ordered links are grouped into a menu. The menu can be structured in a hierarchy. Each link points to a new screen. The target screen contains the menu and possible sub-menus.	
Navigational patterns	wizard pattern [74][75]	A set of triggers are grouped to present the screens in a sequential order. A trigger leads to a subsequent screen, which contains links or buttons labeled with “next” or “continue” to guide a user to navigate through screens.	
Behavioral patterns	multi-value input form pattern, browse pattern, search pattern[68]	A set of widgets in the same screen are grouped together to deliver a single functional unit, such as search, browse or input.	<p>Multi-value Input Form Pattern Browse Pattern</p>

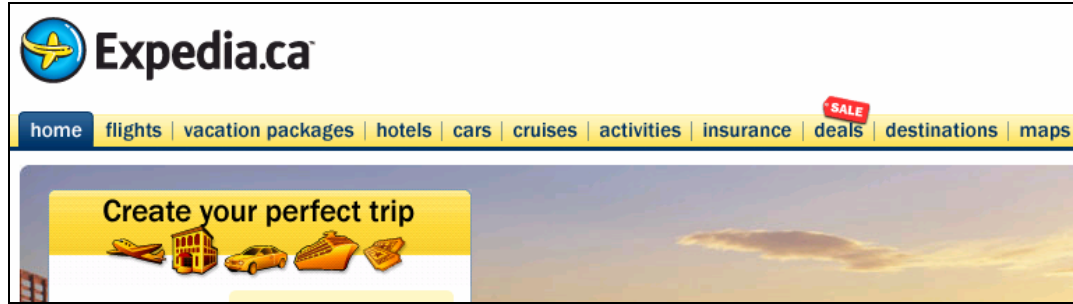


Figure 4-4. Screenshot of a main menu pattern instance [25]

Structural patterns: organize the overall structure of a UI screen by clustering different functionalities into widget groups. The features of structural patterns are presented in Table 4-1. Essentially, the menu provides multiple links, each of which connects to a screen with a functional group (e.g., agreement, accounting, and catalog). An example structural pattern is the main-menu pattern [74][77], as shown in Figure 4-4 from Expedia business application [25]. This pattern is used to display a main-menu which appears at the top of each screen and permits quickly switching among different functionality groups. Structural patterns can contain menus in multiple levels.

To detect structural patterns, we follow the hierarchical relations in the UI widgets and identify the identical link groups appearing in the same level of hierarchy. We examine two features of link groups: each link group must appear in all connected screens; all links in a group are arranged following the same order at a hierarchical level of UI menus. Such a link group is regarded as a menu. By traversing hierarchical structure of the UI menus, we identify the link group for each level of hierarchy. The traversal stops when the link group directly presents UI widgets for performing a collection of business processes. For example, as shown in Figure 4-3(a), the content management application contains a set of links (e.g., Find, Content and Role). The links in this level initiate the UI widgets for conducting the first task, such as Find existing content.

Book flights

1 Search Flight | 2 Select date | 3 Select time | 4 Your details | 5 Review and Pay

Price ticket(s) excluding tax(es) and surcharges: CNY 8080

Price ticket(s) including tax(es), surcharges and reservation fee: CNY 14411

Price based on: 1x adult(s)

Your choice so far:

Departure	Miles earned	Changes possible?	Cancellations possible?	Upgrade with miles
Take Off Fare	25%	X	X	X
Return	Miles earned	Changes possible?	Cancellations possible?	Upgrade with miles
Take Off Fare	25%	X	X	X

Show details and price breakdown

Back Continue to select time

Figure 4-5. Screenshot of a wizard pattern instance [43]

Navigational patterns: describe the navigational structure of the screens. A navigational pattern guides users through screens, as they progress through a process. A business process can be implemented within one screen or can be accomplished across multiple screens. An example navigational pattern is a wizard pattern [74][77] that provides step-by-step instructions to help users complete a complex process. As shown in Figure 4-5, a wizard pattern from the KLM online flights booking application [43] guides users to go through steps (e.g., select date and select time) for purchasing a plan ticket. A navigational pattern indicates the progress of the work and uses “back” or “continue” links to move back for fixing problems or to move forward once the current step is completed. We consider the tasks fulfilled by a navigational pattern as a business process.

Similar to the features of structural patterns, a navigational pattern includes a set of text labels and triggers. The features of navigational patterns are shown in Table 4-1. The text labels

Figure 4-6. Screenshot of a multiple-value input form pattern instance [25]

display a sequence of steps. In contrast to the links in the structural patterns, the text labels are not clickable since the labels are not associated with screens. These text labels often start with sequential numbers which indicate the sequences of steps. A subsequent screen is connected by the “back” or “continue” triggers on each screen. The matching process starts by locating the target screen linked from the “continue” trigger and checks if the target screen contains the same set of labels and triggers. The matching process stops when the “continue” trigger is not associated with a new screen.

Behavioral patterns: characterize functional units delivered in a screen. To name a few, Table 4-1 shows the behavioral patterns recognized in the business applications we analyzed:

- 1) a multi-value input form pattern [68] (i.e., an input form), which encapsulates multiple input widgets. As shown in Figure 4-6 from the Expedia business application [25], different input widgets and submit buttons are used together allow users to edit and submit data.
- 2) a browse pattern [68] (i.e., an output form), which shows the output from back-end components. As shown in Figure 4-3(b), a list of business objects (i.e., contents) are printed in tabular format.

- 3) a search pattern [68] which allows a user to specify search criteria, and to review the results of a search. A search pattern is abstracted to include an input form for entering search criteria and an output form for displaying the result. The screen as shown in Figure 4-3(b) exemplifies a search pattern.

To recognize behavioral patterns in one screen, we first map a form into a candidate behavioral pattern. We expand the candidate behavioral patterns by examining the data dependencies between the form and other widgets (e.g., triggers). A behavioral pattern is identified once we establish data flow relations between the UI widgets and a form. We analyze the data dependencies between the widgets and the forms. We group a widget with an input form, if the data passed to the widget is taken from the form; we group a widget with an output form if the data retrieved from the widget is displayed in the output form. For example, the screen, shown in Figure 4-3(a) has an input form which contains a set of the widgets which gather input from the user. The “Update” button in the screen submits the data gathered from the form to the back-end components. A data flow relation is identified between the form and the button. As shown in Figure 4-3(a), we group the form and the submit button into one behavioral pattern (i.e., the multi-value input form pattern).

To ensure that a single unit of functionality is carried in a behavioral pattern, we further analyze the data dependencies among the identified behavioral patterns. If there is data shared among two identified pattern instances, we group one or more instances into one instance at a higher-level abstraction. An input form and an output form often contain common data fields since the output form may display results triggered by the input form. Both forms are grouped as an instance of the search pattern. For example, looking at the screen shown in Figure 4-3(b), the dependences are identified from the text fields in the input form and the columns in the output form. The combination of these two forms is an instance of a search pattern. The result of the search is listed in the output form.

4.2 Identifying Tasks

After we locate the starting screen of a process through detecting structural patterns, we determine the controller file which describes the connected screens and the invoked back-end components from that screen. We use navigational patterns and triggers in each screen to trace the progress of a business process across multiple screen files and their corresponding controller scripts. This tracing process continues until we re-visit the starting screen or there are no new triggers to visit. We use the title of the initial screen to name the recovered business process.

To identify tasks, we apply the following three steps in each UI screen:

- 1) Identify tasks by recognizing the behavioral patterns (e.g., input or output form) in each screen. Each identified behavioral pattern is mapped to a task.
- 2) Name the tasks using the name of the form. When a task is recovered from a behavioral pattern that is composed of several forms, the name of a recovered task is prefixed using the name of the behavioral pattern (e.g., input, browse or search).
- 3) Identify a group of tasks as a reusable sub-process. A behavioral pattern may contain buttons that invoke back-end components. The detailed steps conducted in the back-end components are resolved when recovering the corresponding low-level processes with the business process recovery techniques in [35][36]. The button is, therefore, recovered as a sub-process which corresponds to a back-end component. One screen in the UI can be reused in different processes. When more than one task is recovered from the reusable screen, we capture the tasks recovered from the screen into a sub-process. Such a sub-process is reused as a single unit in other recovered processes when the same screen is used.

For example as shown in the “Edit Content” screen of Figure 4-3(a), the input form is recovered as “Edit Content Attributes” task. The update button invokes a back-end component

that takes all the data entered in the input form. The portion of the high-level process recovered from the screen is depicted in Figure 4-7(a). The “Update Content” sub-process encapsulates the detailed steps carried in the back-end component as shown in Figure 4-7(b). We recover sub-processes from back-end components through data flow tracing and program slicing techniques. The sub-process in Figure 4-7(b) delivers the functionality to update content. The first two tasks (i.e., Prepare Target Operation List and Prepare Content Purpose List) prepare related business data for following operations. After checking the permission status, the changes of content will be stored into database. There will be an error message displayed, if the update content operation is not permitted.

When no behavioral patterns are identified in a screen, a task is recovered from a link connected to a new screen or a button which invokes a back-end component. When a screen has no recovered task, the link, connecting to the screen, is simply mapped to a task in the recovered process.

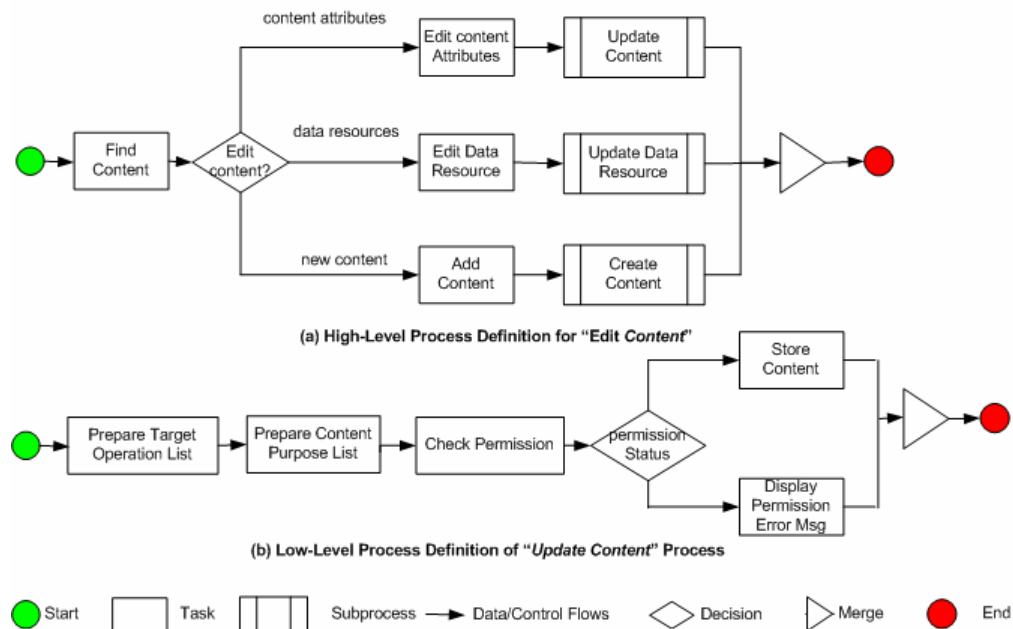


Figure 4-7. Example high-level process vs. low-level process

4.3 Identifying Control Flows

In addition to recovering the tasks or sub-processes, we must identify the interaction and coordination between these tasks or sub-processes. The tasks (or sub-processes) are coordinated by control flow constructs such as sequence, loop, and alternative. We currently cannot identify parallel control constructs.

Sequence: To reduce the number of clicks in a screen and avoid frequently transferring small amounts of information over the network, developers group a sequence of tasks into one screen. Once all the tasks in the screen are completed, a new screen is generated for performing the next step. Therefore, the sequential order of the tasks recovered within one screen is derived from the ordering of the behavioral patterns within this screen. Moreover, the data dependencies among the tasks (or sub-processes) convey the sequential order.

To locate the sequential order of the tasks conducted among multiple screens, we recognize whether navigational patterns are used to move between screens. The order of the links presented in a navigational pattern indicates the order of executing tasks (or/and sub-processes) recovered from each screen. When no navigational patterns are detected in the screens, the order of screen transitions determines the sequential order of the tasks recovered from each screen.

Alternative: Multiple triggers can be selected in one screen. Once one of the triggers is selected, a new screen is generated to replace the previous screen without completing other widgets in the screen. The triggers are independent without data flowing among them. Such triggers in a screen present alternative relations. For example in the screen shown in Figure 4-3(a), the four triggers (i.e., the three links or click on the “Update” button) in this screen are in an alternative relation. To locate alternative relations, we examine the triggers in the same screen without data dependencies. The tasks (or sub-processes) recovered from the independent triggers

are in alternative relations. We ignore the triggers that link to the first screen and restart the entire process.

Loop: If a screen other than the starting screen for a process is revisited, a user is required to repeat the tasks that were completed until the repetition is terminated by the user. The sequence of tasks being repeated is encapsulated in a loop control flow construct in the recovered process. Different from [36], we identify the revisiting of the starting screen of a process as a restarting of the whole process rather than a loop control flow.

Chapter 5

Clone Detection in Business Processes

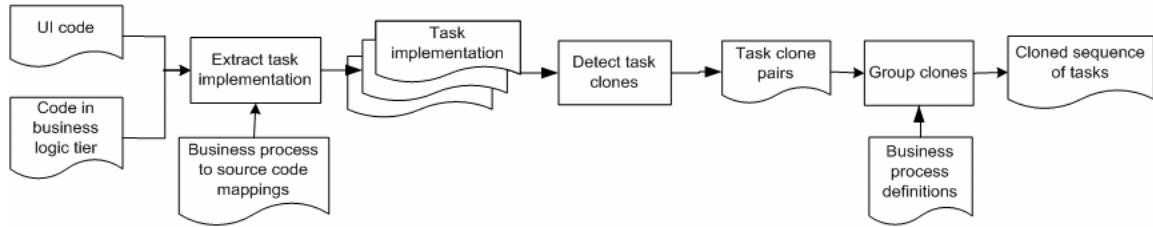


Figure 5-1. Steps for detecting task clones in business process definitions

The overall steps for identifying task clones are illustrated in Figure 5-1. Our approach analyzes the implementation of a business application and extracts the implementation of each task by using the business process to source code mapping information, which is recorded when we recover business processes. Once the implementation of each business task is extracted, we perform clone detection on these tasks using different clone detection tools. We define a metric for measuring the similarity between tasks according to the common code fragments in their corresponding implementation. Finally, we lift and abstract the identified clones by recognizing the usage patterns of groups of task clones across multiple processes. Using our recovered clones, we can improve the definition of business processes by establishing references between cloned tasks across business processes. In the following subsections, we discuss the details for each step.

5.1 Detecting Clones from UI Tiers

A web-based UI page can be described using markup languages (e.g., HTML and XML) and scripting languages (e.g., JSP and JavaScript). A UI page contains a set of nested tag structures. Each tag includes attributes and values. For example, as shown in Figure 5-2, the tag, *Form*, has a set of attributes, such as *method* and *action*. For the dynamic web-UIs, scripting languages, such as JSP and JavaScript, are embedded in the pre-defined HTML tags.

**Figure 5-2. Pretty-print UI code**

To recover task clones among tasks mapped to UI code, we extract the related code of these tasks using the mapping information (e.g., line numbers and XPath expressions for XML elements) stored in the process definitions. As a consequence, the code clone detection is applied only on the business relevant code rather than on the entire UI code. The linked code fragments can contain one or more tags (e.g., tables, hyper links or forms). Similar to the approach used in [17], we pretty-print the extracted UI code fragments to a consistent format which makes each element (e.g., tag, each attribute of a tag, and text contents) take a single line. The pretty-printing helps the clone detection tool in detecting cloned tags and attributes. As shown in Figure 5-2, the pretty-printing transforms a Form tag and an Input HTML tag and the associated attributes into a set of lines. We remove the comments and indentations in the code fragments.

After we extract and pretty-print code fragments corresponding to one task, we merge these code fragments contributing to one task into a single file. The code clone detection is conducted on the generated files. The extracted code fragments may contain code written in scripting languages (e.g., JSP). We select a token-based clone detector CCFinder [41] and set it in the 'PlainText' mode [12] to detect clones from source files written in different languages.

We develop a tool that parses clone results produced from CCFinder to obtain a set of clone groups which collect the code fragments cloned with each other. The code fragments in a clone group can be a subset of tags, attributes and text contents belonging to a task implementation. Furthermore, a task implementation may contain multiple clones from different groups.

5.2 Detecting Clones from Business Logic Tiers

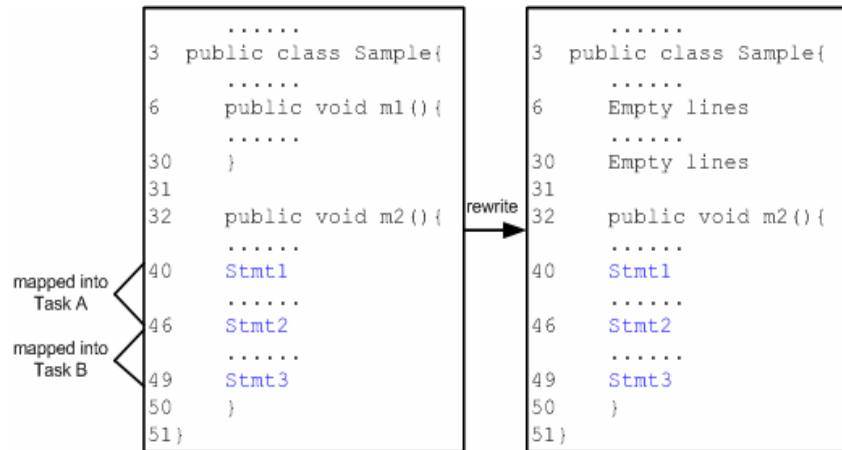


Figure 5-3. Rewrite code fragments from a class file

A process definition records the line number of each statement contributing to a task recovered from the business logic tier. We focus on detecting clones in the business relevant code and ignore the code with implementation details. However, a task implementation contains only a subset of the code in a method. To preserve the control and data dependencies of the code in the task implementation with the rest of the code in a method, we extract the entire methods that have one or more lines of code contributing to a task implementation, and group such methods into a separate class file. Once clones are identified from the extracted files, we must map the cloned code fragments to the corresponding tasks. The mappings are established using the line numbers of the statements in the original source files. To keep the line numbers of each extracted statement unchanged in the new files, we replace the other irrelevant methods with empty lines. This can guarantee that the original lines of code stored in the process definitions are still valid to

map to the task implementations. For example in Figure 5-3, the method, *m2()*, is copied into the new file since it contains one or more statements mapped to task implementations. The method, *m1()*, is replaced with empty lines since no code fragment is related to task implementation in *m1()*. The clone detection is conducted on the extracted files.

We use CloneDR [15], an abstract syntax tree based clone detector for identifying clones in the business logic tier. We develop a tool to automatically parse the clone results and obtain a set of clone groups which contain identical or almost identical code clones. The cloned statements contributing to a task can be distributed among several clone groups.

For detecting clones in the UI tier, we use a token based clone detection technique, CCFinder [41], since the UI tier is usually written in various languages for which we may not have the grammar definitions. The grammar definitions are needed for abstract syntax tree based techniques. We use an abstract syntax tree based technique, CloneDR, for detecting clones in the business logic tier, since abstract syntax tree based techniques are known to be more accurate than token based techniques for detecting clones in source code [39].

5.3 Measuring Similarity between Tasks

Once we detect code clones from code fragments mapped into tasks, we measure the similarity between tasks to identify task clones. The similarity between two tasks is denoted by the ratio of common lines of code between both tasks. The common lines include the cloned lines in both task implementations and the overlapped fragments shared by several task implementations. For example as shown in Figure 5-3, line 46 is the overlapped line that belongs to two task implementations located in the same file. When the cloned tasks are detected from one file, the overlapped lines among the task implementations have only one copy and are not recognized as cloned lines. The similarity is calculated by Eq. 1.

$$Sim(T_1, T_2) = \frac{\text{Cloned Lines} + \text{Cloned Overlapped Lines}}{\text{Total Number of Lines in } T_1 \text{ and } T_2} \quad (\text{Eq. 1})$$

$Sim(T_1, T_2)$ denotes the similarity between tasks T_1 and T_2 .

When the similarity among tasks is above a certain threshold, we recognize the task pairs as task clones of each other. For tasks recovered from the UI tier, we calculate the similarity between two task implementations in term of common XML or HTML elements. The threshold is set to be 50% or 60%. We chose this value after we conducted a series of experiment using thresholds ranging from 40% to 70%. For the tasks recovered from the business logic tier, the threshold is 30%. This value is selected after a series of experiments with thresholds from 20% to 40%. Although the value 30% seems to be low, it is still a strict criterion. For tasks recovered from the business logic tier, we calculate similarity between task implementations, where the seed statements are clones. In particular, the seed statements describe the basic functionalities delivered by business tasks. The cloned seed statements deliver similar functionality. The selected thresholds avoid most of the misidentified cloned tasks.

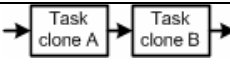
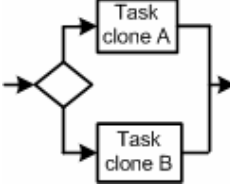
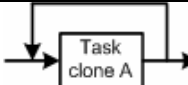
5.4 Grouping Task Clones

The cloned task pairs appear in multiple processes to represent the unique functionality. To help business analysts identify larger clone segments, we further lift the abstraction level of business processes by identifying clones among a group of tasks. A preliminary list of clone structures among tasks is summarized in Table 5-1. A group of tasks may appear in a process in three possible control structures: sequential task clones, alternative task clones and loop task clones.

- Sequential task clones: a task, A, and the clones of task A are frequently used together with task B or the clones of task B.

- Alternative task clones: a task, A, and the clones of task A are frequently in alternative relation with task B or the clones of task B.
- Loop task clones: the sequential task clones are used in a loop structure.

Table 5-1. Preliminary list of structures for grouping tasks

Task structure	Process diagram	Explanation
Sequential task clones		Task clone A and its clones appear together with task clone B and clones of B in sequential order.
Alternative task clones		Task clone A and its clones appear together with task clone B and clones of B in different control branches.
Loop task clones		Task clone A and its clones appear together in a loop.

When a group of task clones is repeatedly used together in different processes, such a group represents a reusable unit (i.e., a sub-process). The sub-process uses the same control flow constructs to connect cloned tasks or the same tasks. To lift the clone from the task level to the process level, we identify the identical control structures that describe the same execution order of cloned tasks in different processes. We consolidate the tasks and the same control structures as a sub-process and replace original tasks with a single notation of the extracted sub-process. Once a sub-process is replaced, it can be treated as a task to further identify task clone groups and lift the abstraction level of clones into a larger scope.

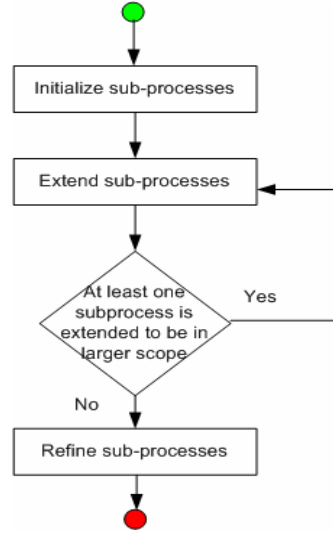


Figure 5-4. Steps for extracting sub-processes

As shown in Figure 5-4, extracting a sub-process consists of three major steps: initialize sub-processes, extend sub-processes and refine sub-processes. The details of these steps are presented in algorithms listed in Figures 5-7 to 5-13. Figure 5-5 is an example about our approach to extract sub-processes.

Figure 5-7 describes the *Initialize sub-processes* algorithm that parses process definitions and initializes sub-processes by grouping identical or cloned tasks into a sub-process. For example, as shown in Figure 5-5(a), tasks T_1 and T_1' are cloned tasks; task T_2 and T_2' are cloned tasks; task T_3 appears in processes 1 and 2. We parse processes 1 and 2, and group tasks to produce the initial sub-processes as shown in Figure 5-5(b). Each sub-process contains multiple instances, each of which contains a set of nodes (e.g., task nodes) appearing together.

Figure 5-8 depicts the *Extend sub-processes* algorithm that extends sub-processes into larger scopes by grouping sequent neighbor nodes (Figure 5-9), and alternative neighbor nodes (Figure 5-10), and recognizing identical control structures (Figure 5-11). We continue to expand sub-processes until no more new sub-processes are found.

We define *extendIncludingSequentNeighbors* algorithm as listed in Figure 5-9 to extend a sub-process by grouping identical or cloned tasks in the sequential relations. For a task

coordinated by a sequence control flow construct, the sequential tasks are ones following the sequence control flow construct. As shown in Figure 5-5(a), T_2 and T_2' are the cloned subsequent tasks of T_1 and T_1' in sub-process 1 of Figure 5-5(b). Therefore, we extend sub-process 1 to include T_2 and T_2' as shown in Figure 5-5(c).

The *extendIncludingIdenticalControlStructures* algorithm shown in Figure 5-11 describes the steps to group identical control structures. Two control flows are identical when they use the same type of control flow constructs to coordinate cloned or identical tasks. Applying the algorithm (Figure 5-11) on sub-process 1 in Figure 5-5(c), we can update sub-process 1 to contain two identical sequent control structures respectively including T_1 and T_2 as well as T_1' and T_2' as shown in Figure 5-5(d).

We can also expand sub-processes by grouping cloned or identical alternative neighbor tasks, as described in the *extendIncludingAlternativeNeighbors* algorithm of Figure 5-10. A task coordinated by an alternative control flow construct is the neighbor of other tasks controlled by the same structure. From the initial processes in Figure 5-5(a), we can tell the two T_3 task nodes are alternative neighbor nodes of the two sequent control structures in sub-process 1 of Figure 5-5(d). Therefore, as shown in Figure 5-5(e) we extend sub-process 1 to include the identical tasks T_3 . Then, using algorithm *extendIncludingIdenticalControlStructures* (Figure 5-11), we can update the sub-process 1 to contain identical alternative control structures as shown in Figure 5-5(f).

After we extend a sub-process, we use the *mergeIntoSubprocessList* algorithm in Figure 5-12 to merge the extended sub-process into a list to guarantee there are not duplicated sub-processes. When no more sub-processes can be found, we use the *refineSubprocesses* algorithm in Figure 5-13 to create sub-process definitions and replace tasks in original processes with the notation of the extracted sub-processes. As shown in Figure 5-5(g) the original tasks are replaced with the extracted sub-process 1.

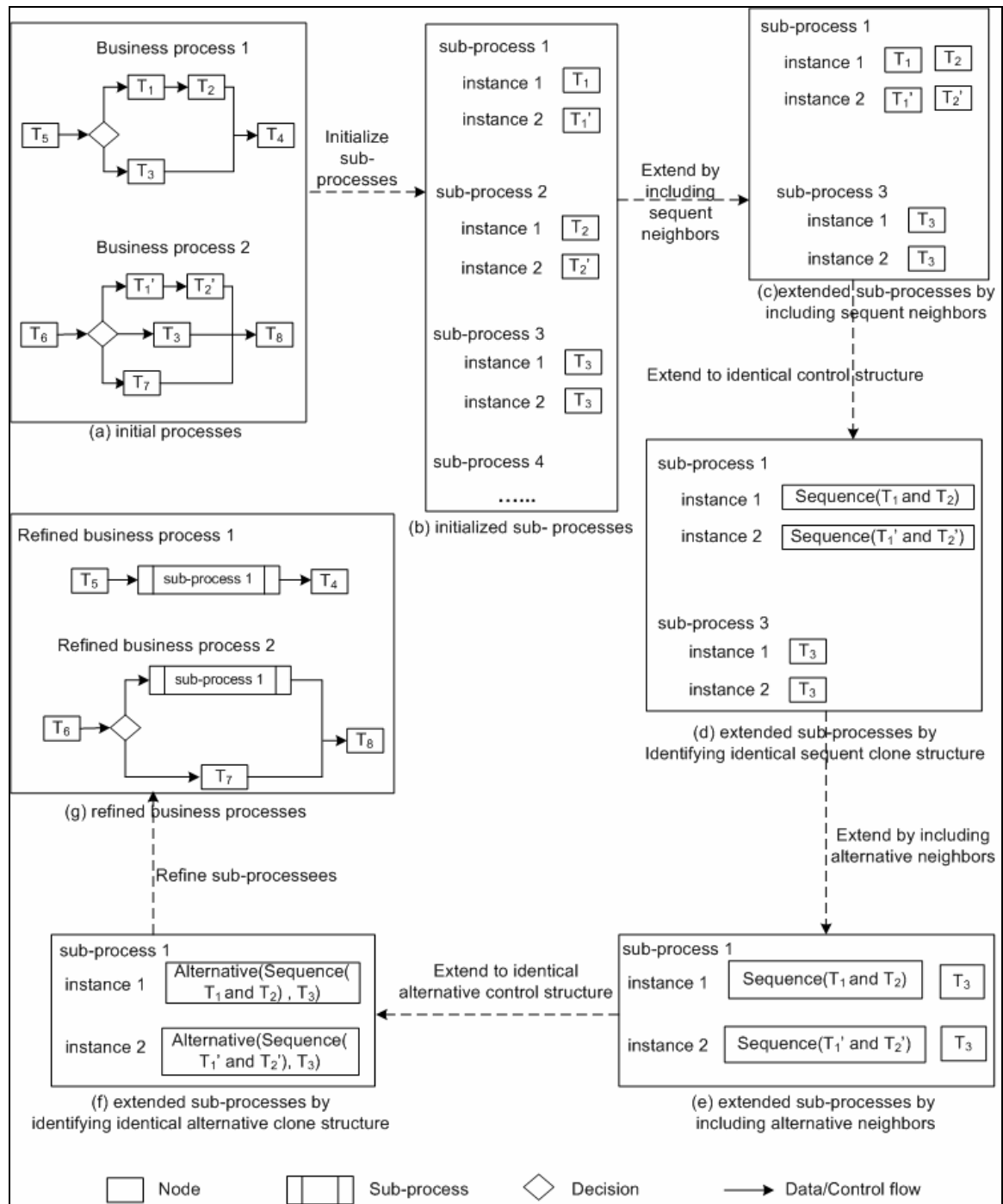


Figure 5-5. An example of extracting sub-processes

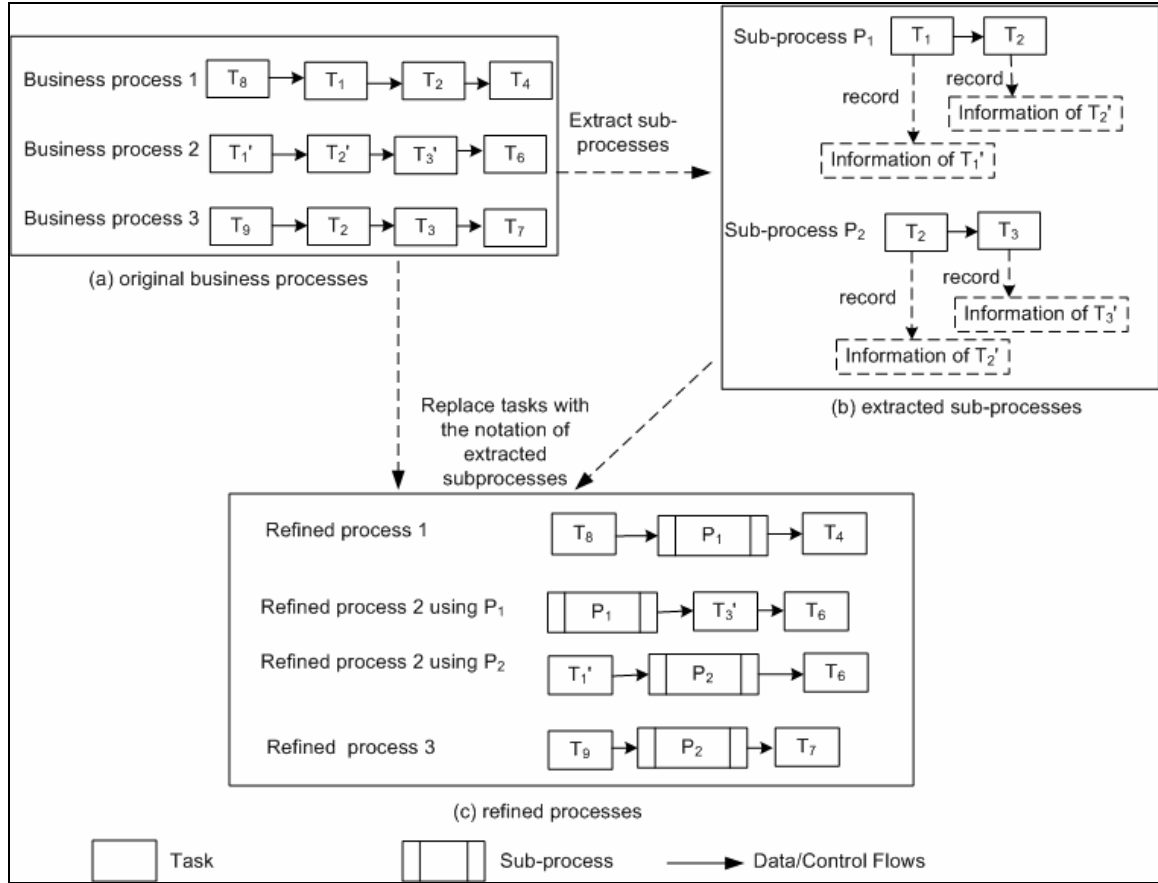


Figure 5-6. An example for refining processes

When we extract sub-processes, it is possible for the extracted sub-processes to have overlapping. That means a task node or a control structure (e.g., sequence node) are able to fall in different sub-processes. In the three processes of Figure 5-6(a), tasks T_1 and T_1' are task clones; tasks T_2 and T_2' are task clones; tasks T_3 and T_3' are task clones. From the three processes, we are able to detect two sub-processes P_1 and P_2 as shown in Figure 5-6 (b). Sub-process P_1 appears in processes 1 and 2, and contains sequence control structures including tasks (T_1, T_2) and tasks (T_1', T_2'). Sub-process P_2 appears in processes 2 and 3, and contains sequence control structures including tasks (T_2, T_3) and tasks (T_2', T_3'). There is overlapping between P_1 and P_2 . As shown in Figure 5-6 (c), when we refine sub-processes by replacing tasks with a notation of the extracted sub-process, process 2 has two different refined copies respectively using P_1 and P_2 .

Algorithm 1. Initialize sub-processes

Input: *processDefinitions*: is a list of process definitions from which we want to extract sub-processes.
Return: *subprocessList* is a list of sub-processes. (Each sub-process contains a list of instances; each instance has a set of nodes appear together)

Parse *processDefinitions*

Cluster task nodes to put cloned task nodes and identical task nodes into different instances of a sub-process

Put the sub-process with more than one instance into *subprocessList*

return *subprocessList*

Figure 5-7. Algorithm for initializing sub-processes

Algorithm 2. Extend sub-processes

//extended is set to be true if at least one subprocess is successfully extended

extended = false;

for each *subprocess* in *subprocessList* **do**

 //extend by including sequent nodes (i.e., nodes nested by Sequence node)

call algorithm 3: *extendIncludingSequentNeighbors*

 //extend by including alternative nodes (i.e., nodes nested by Alternative node)

call algorithm 4: *extendIncludingAlternativeNeighbors*

 //extend by including identical control structures (i.e., Sequence nodes)

call algorithm 5: *extendIncludingIdenticalControlStructures*

 set *extended* to be true if at least one of above algorithms return true

end for

return *extended*

Figure 5-8. Algorithm for extending sub-processes

Algorithm 3. extendIncludingSequentNeighbors

Input: *subprocess* is a sub-process in *subprocessList*

Return: *extended* to reflect is there any new subprocess created

//extended will be true if at least a new sub-process is successfully create

extended = false;

copiedSubprocess = copy *subprocess*

//a instance contains a list of nodes (task nodes and control structures) appearing together

Remove instances, which don't contain nodes coordinated by sequence control flows, from *copiedSubprocess*

Extend instances in *copiedSubprocess* to include identical or cloned sequent neighbor nodes of the last nodes in instances, until no more equal nodes can be found.

extendedSubprocess = split *copiedSubprocess* into a list of sub-processes, each of which contains at least two similar instances

//merge all extended sub-processes into list and avoid to insert duplicated sub processes

extended = call algorithm 7: mergeIntoSubprocessList (*extendedSubprocesses*)

return *extended*

Figure 5-9. Algorithm for extending sub-processes by including sequent neighbor nodes

Algorithm 4. extendIncludingAlternativeNeighbors

Input: *subprocess* is a sub-process in *subprocessList*

Return: *extended* to reflect is there any new subprocess created

//extended will be true if at least a new subprocess is successfully created

extended = false;

copiedSubprocess = copy *subprocess*

Remove instances, which don't contain nodes coordinated by alternative control flows, from *copiedSubprocess*

Extend instances in *copiedSubprocess* to include new identical or cloned alternative neighbor nodes, until no more nodes can be found.

extendedSubprocess = split *copiedSubprocess* into a list of sub-processes, each of which contains at least two similar instances

extended = call algorithm 7: mergeIntoSubprocessList (*extendedSubprocesses*)

return *extended*

Figure 5-10. Algorithm for extending sub-processes by including alternative neighbor nodes

```

Algorithm 5. extendIncludingIdenticalControlStructures
Input: subprocess is a sub-process in subprocessList
Return: extended to reflect is there any new sub-process created

//extended will be true if at least a new sub-process is successfully created
extended = false
copiedSubprocess = copy subprocess

//a instance contains a list of nodes (task nodes and control structures)
for each instance in copiedSubprocess do
    if nodes in instance are coordinated by a sequent or loop control structure do
        if nodes contain all the children nodes under this control structure do
            update instance to include nodes and the control structure
        else
            remove instance
        end if
    else if nodes in instance are coordinate by alternative control structure
        if nodes contain at least two nodes coordinated by the alternative control structure do
            update instance to include nodes and the alternative control structure
        else
            remove instance
        end if
    end if
end for

extendedSubprocesses = split copiedSubprocess into a list of sub-processes, each of which
contains at least two similar instances

extended = call algorithm 6: mergeIntoSubprocessList (extendedSubprocesses)
return extended

```

Figure 5-11. Algorithm for extending sub-process by including identical control structures

Algorithm 6. mergeIntoSubprocessList

Input: *extendedSubprocesses* is a list of newly extended sub-process

Return: *inserted* to reflect is there any new sub-process inserted

```

extended = false
for each subprocess in extendedSubprocesses do
    if the subprocess is not existed in subprocessList and subprocess is not a subset of any element
    in subprocessList then
        //subprocessList is a list of all extracted sub-processes
        Insert subprocess into subprocessList

        // avoid duplicated subprocesses
        Remove the elements in subprocessList which is a subset of the inserted subprocess

        //record that a new sub-process is inserted
        extended = true
    end if
end for
return extended

```

Figure 5-12. Algorithm for merging extended sub-processes into list

Algorithm 7. refineSubprocesses

Input: *subprocessList* is the list of extracted sub-processes

```

for each subprocess in subprocessList do
    cloned = false
    for each instance in subprocess do
        for each node in the instance do
            if the node has cloned task appear in similar position of other instances then
                Insert the cloned information into node
                cloned = true
            end for
        end for
        //make sure the sub-process contains at least one group of task clones
        if cloned equals to false then
            Remove subprocess from subprocessList
        else
            Extract all instances in the subprocess into a process definition
            Replace the appearance of all instances with a single notation of the extracted process
            definition
        end if
    end for

```

Figure 5-13. Algorithm for refining sub-processes

Chapter 6

Overview of Business Process Explorer Tool

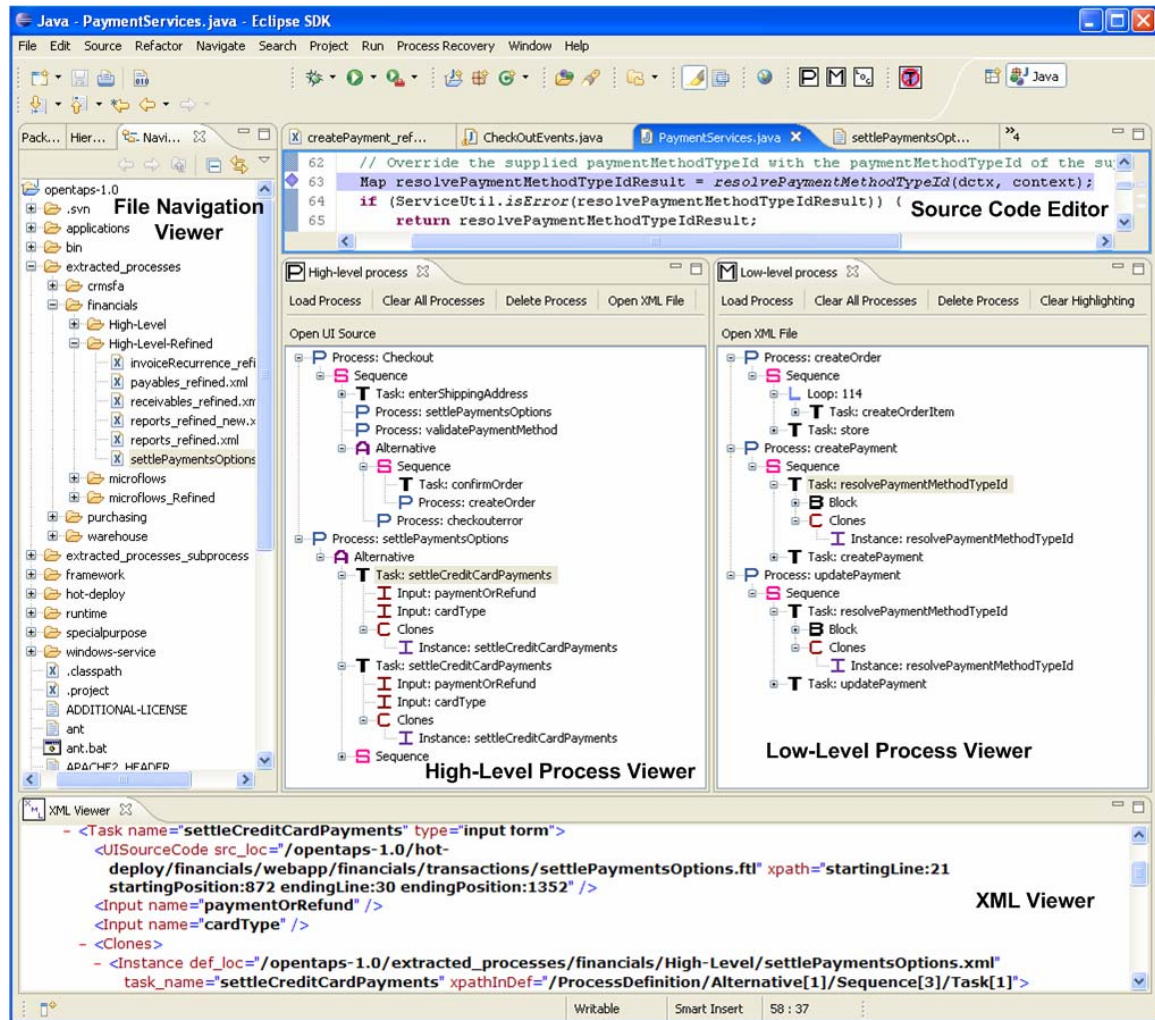


Figure 6-1. An annotated screenshot of the business process explore

We developed a business process explorer (BPE) tool [30] which automatically recovers business processes from business applications and detects task clones in recovered business processes. An annotated screenshot of the BPE tool inside Eclipse [22] is shown in Figure 6-1. The code of the business application is imported into the BPE tool and listed in the Eclipse file navigation viewer. The recovered business processes are also stored in file navigation viewer and defined as XML documents.

To help software developers quickly locate the code blocks that implement a task, our tool maintains the traceability between process entities (e.g., tasks and control flows) and the code entities (e.g., code blocks). We represent the recovered business processes with two levels of abstraction: high-level and low-level processes. As shown in Figure 6-1, the high-level process viewer displays tasks recovered from the UI code and sub-processes. A sub-process encapsulates a sequence of tasks to be reused in other processes. The low-level process viewer lists the sub-processes and tasks recovered from the business logic tier. Once a process entity in the low-level process viewer is selected, the corresponding task implementation is highlighted in the source code editor. As shown in the low-level process viewer of Figure 6-1, the tasks for “Create order” are listed. When a user double clicks on the task node in the low-level process viewer, the lines of code corresponding to the clicked task are automatically loaded and highlighted in the source code editor.

To provide a hierarchical view of business processes, as shown in Figure 6-1, we represent the process as a tree structure in the viewers (e.g., high-level process viewer and low-level process viewer). Nodes annotated with different icons in the trees represent various process elements, including control flow elements (i.e., alternative, choices, and sequence), tasks and business data as parameters to tasks. The nodes labelled as “Task” represent the human tasks performed by a user through the UI or a task executed in back-end components. The nodes labelled as “Process” represent sub-processes recovered from the invoked back-end components or UI screens. Once a user clicks on one of the “Process” nodes in the high-level process viewer, the corresponding sub-process is loaded. When the sub-process represents a low-level process, the sub-process is viewed as a separate process using a similar tree structure in the low-level process viewer. When the sub-process is recovered from a UI screen, the sub-process is listed in the high-level process viewer.

The recovered business process is represented using XML format as shown in the XML viewer in Figure 6-1. A developer can directly edit the XML representation of a process to refine a recovered process.

Our business process explorer can identify task clones existing in recovered business processes by parsing the code clones provided by clone detection tools (i.e., CCFinder and CloneDR) and raising the clone from source code level to business process level. After task clones are identified, the business process definitions will be refined to explicitly denote task clones and suggest restructuring opportunities. As shown in Figure 6-1, in high-level and low-level process viewers, we use “Clones” nodes to represent the clone information, and use an “Instance” node to represent a process entity (e.g., task and process) which is similar to another entity. In the process “settlePaymentsOptions” loaded in high-level process viewer, there are two tasks “settleCreditCardPayments” which are task clones of each other. When we highlight an “Instance” node, and click the “Open UI Source button”, the UI file that implements this cloned task will be presented in source code editor. In the low-level process viewer, the two “resolvePaymentMethodTypeId” tasks located in processes “createPayment” and “updatePayment” are task clones of each other. When we double click an “Instance” node, the corresponding java code implementing this cloned task will be high-lighted in source code editor.

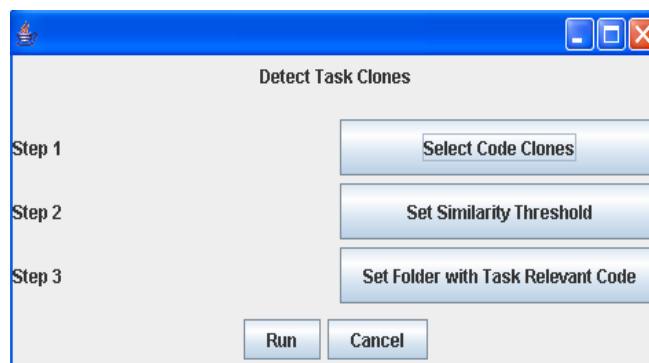


Figure 6-2. Steps for detecting task clones from high-level processes

Figure 6-2 is a screenshot of the wizard dialog that guides users to detect task clones in high-level business processes. In the first step, users select code clone results provided by CCFinder that is applied to UI task relevant codes (i.e., UI code blocks mapped into tasks); in the second step, users set a similarity threshold value between 0 and 1 to filter out task clones; in the third step, users select the folder storing UI task relevant source code. Tasks with similarity higher than the selected threshold will be identified as task clones.

Figure 6-3 shows steps to detect task clones in low-level processes. Users select the code clone results provided by CloneDR, which is applied on task relevant codes located in back-end components, and set a similarity threshold value between 0 and 1. Since task relevant codes extracted from back-end components are stored under the project folder of the studied business application, users don't need to specify the location of task relevant codes.

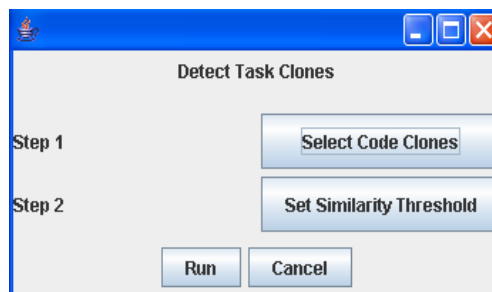


Figure 6-3. Steps for detecting task clones from low-level processes

To ease the adoption of recovered processes in practice, we developed a converter to automatically transform the process definitions from our proprietary schema to the XML schema recognized by the IBM WBM. Our recovered processes can be visualized in WBM. As shown in Figure 6-4, a business process recovered by the business process explorer is being visualized in WBM. Each element type in the process is represented by a different shape in the visual editor. Properties of each element can be modified by adjusting the values in the property editor. In the project navigator illustrated in Figure 6-4, a user can view the list of business data currently defined in the process. The visualization allows a business analyst to navigate through the

recovered processes and to give more meaningful names to the tasks in recovered processes. Using WBM, business processes implemented in the source code can be kept up-to-date. We can also leverage the powerful features provided by WBM to analyze the recovered processes. For example, WBM provides a simulation tool that allows the user to simulate the execution of a process to locate possible design problems. Consequently, the process can be optimized to improve the performance of the organization which owns the e-commerce applications.

This business process explorer tool is developed by members in Software Reengineering Research Group at Queen's University under the supervision of Professor Ying Zou. I developed the components that recover business processes from user interfaces and detect task clones.

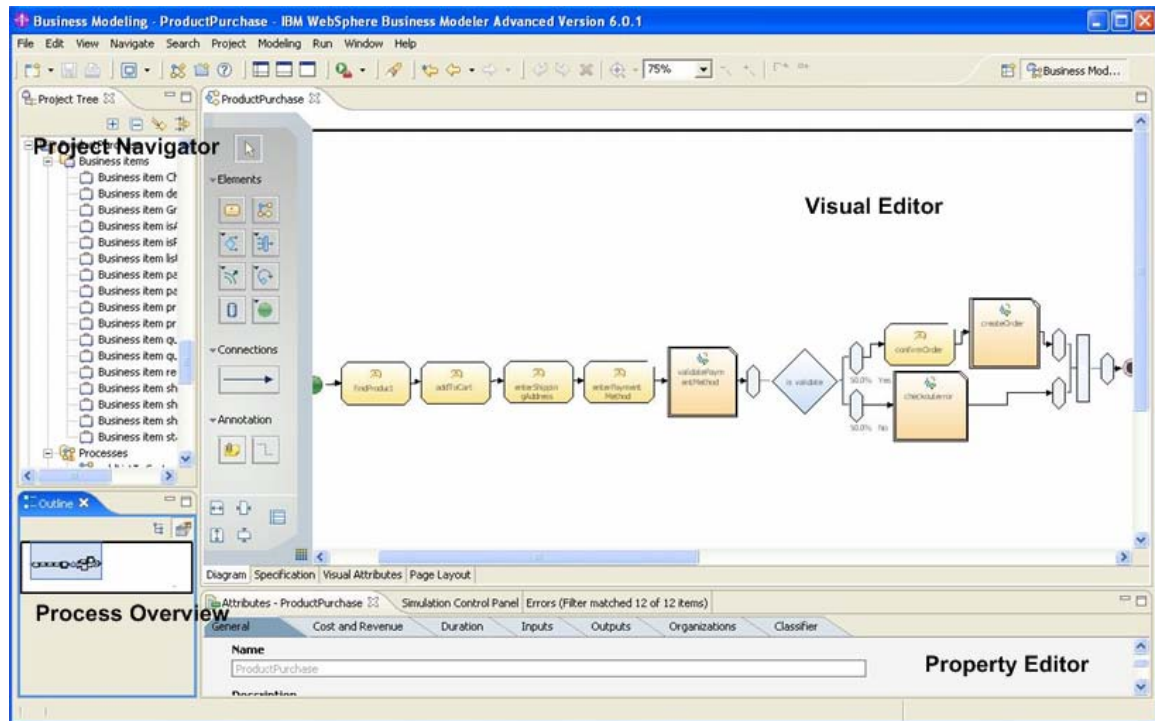


Figure 6-4. Visualizing recovered processes in IBM WBM

Chapter 7

Case Studies

This chapter presents the case studies to evaluate the effectiveness of our business process recovery and task clone detection techniques.

To validate our proposed techniques, we applied business process explorer tool [30] to recover business processes from the studied applications and manually verified the recovered business processes. Since this thesis focuses on recovering high-level business processes starting from UIs, we only analyzed the results recovered from UI screens. Moreover, we applied task clone detection and process refinement techniques on the recovered business processes. With the help of the business process explorer tool, we established the mappings between business processes and source code in the UI and business logic tiers. We applied CCFinder to detect code clones in the UI tier, and CloneDR to locate code clones in the business logic tier. Furthermore, we developed a prototype tool, which has been integrated into the business process explorer, to analyze the results produced from the clone detection tools to identify task clones. Our tool consolidated repeated task groups into sub-processes and refined the process definitions. Finally, we manually verified the detected task clones and the refined business process definitions.

Section 7.1 introduces the characteristics of the 15 case study applications from 3 large open source projects. Section 7.2 describes the evaluation of our business processes recovery approach. Section 7.3 discusses the evaluation of the task clone detection approach.

7.1 Application Studied

We used 15 business applications in our case study. The applications come from three open source projects: OFBIZ SequoiaERP [66], OFBIZ Opentaps [58], and SMaCS[69]. Table 7-1 lists the characteristics of the studied applications.

Table 7-1. Characteristics of the subject applications

Projects	Application	User interface trier		Business logic trier			Database trier
		Language	Number of parsed files	Language	Number of classes	Line of code	Platform
SequoiaERP (0.8.0)	Accounting	XML, FTL, JSP	54	Java, XML minilang	19	8240	Data beans
	Catalog	XML, FTL, BSH	136	Java, XML minilang	34	10283	Data beans
	Facility	XML, FTL, BSH	87				
	Content	XML, FTL, BSH	204	Java, XML minilang	41	10178	Data beans
	Ecommerce	BSH, FTL	165	Java, XML minilang	1	145	Data beans
	Manufacturing	XML, BSH	77	Java, XML minilang	12	3622	Data beans
	Marketing	JSP, XML	15	Java, XML minilang	1	231	Data beans
	Order	XML, BSH, FTL	127	Java, XML minilang	29	13677	Data beans
	Party	XML, FTL, BSH	46	Java, XML minilang	8	2330	Data beans
	Workeffort	XML, BSH	50	Java, XML minilang	5	781	Data beans
Opentaps (1.0.0)	Crmsfa	XML, FTL, BSH	255	Java, XML minilang	37	7535	Data beans
	Warehouse	XML, FTL, BSH	99	Java, XML minilang	11	1595	Data beans
	Purchasing	XML, FTL, BSH	85	Java, XML minilang	8	1085	Data beans
	Financials	XML, FTL, BSH	108	Java, XML minilang	22	5611	Data beans
SMaCs	N/A	HTML, JSP, JavaScript	183	Java	73	5600	SQL

7.1.1 Characteristics of SequoiaERP and Opentaps

Both Opentaps and SequoiaERP are available as commercial projects by Open Source Strategies Inc and as open source projects. The open source projects provide the major functionality of the commercial projects and are developed by the same team of developers. We used the open source projects in our case study. SequoiaERP is an earlier version of Opentaps. Opentaps and SequoiaERP offer a suite of independent applications for ERP (Enterprise Resource Planning), CRM (Customer Relationship Management) and E-Commerce. SequoiaERP contains 10 applications [58]. Opentaps, as a new version of SequoiaERP, has more applications in

addition to the 10 applications in SequoiaERP. The newer versions of the 10 common applications have a better structured UI with fewer links among screens. To demonstrate the strength of our recovery techniques in handling complex UIs, we decided to use the older versions of the applications from SequoiaERP. We also used 4 additional applications from Opentaps. Similar to SequoiaERP, each of the applications in Opentaps is independent from others, and can be used separately. The detailed information for each application in Opentaps and SequoiaERP can be found at [58].

Both projects are built on the Apache Open for Business (OFBIZ) framework. The OFBIZ framework offers a common infrastructure for invoking back-end components and representing the UIs and data models for business applications. The framework defines an XML scripting language, called minilang, which allows developers to easily retrieve and update business data in databases. The screens and controller files are written in XML. The controller files configure the screens and specify the input and output of widgets displayed in screens. The controller files describe the parameters used in the interfaces of the invoked back-end components. The back-end components are implemented using Java or minilang. The data model files specify the business data used in an application. Each application in SquoiaERP and Opentaps is developed by different developers. Although each application must follow the infrastructure defined in the OFBIZ framework, OFBIZ still allows a great degree of flexibility for developers to exhibit their own coding styles. For example, developers can implement a task using different code blocks, or they can choose to implement a task as a single method.

7.1.2 Characteristics of SMaCS

To examine the generality of our approach, we selected SMaCS, another business application which does not depend on the OFBIZ framework. The application helps manage casual staff in an organization. The application has business processes for electronic bookkeeping of rosters and timesheets, and for generating pay schedules. In contrast to the OFBIZ

applications, where the data beans are defined using data model files, SMaCS defines data beans using Java code. The methods in the data beans capture the SQL statements needed to access and update a particular business data stored in a database.

7.2 Business Process Recovery

The following sections will describe the case studies and discuss the results.

7.2.1 Measuring the Effectiveness of Business Process Recovery Techniques

There are two standard information retrieval metrics, precision and recall, used to evaluate the effectiveness of our recovery approach. Precision measures the ability of our approach to identify and exclude non-business relevant tasks from the recovered processes. Recall measures the ability of our approach to identify and include all business relevant tasks in the recovered processes. Eq. 2 measures the precision of our approach for recovering a business process. Eq. 3 measures the recall of our approach for recovering a business process. Misidentified tasks refer to the tasks which our approach identified incorrectly as business relevant tasks. Missed tasks refer to tasks which our approach did not identify. Each application contains several business processes so we reported the precision and recall of all the processes recovered from each application in our case study. We sought an approach with high precision to ensure that the practitioner's time is not wasted examining irrelevant code or tasks, and with high recall to ensure that the practitioner does not miss any business relevant code or tasks.

$$precision = \frac{\# \text{ of identified tasks} - \# \text{ of misidentified tasks}}{\# \text{ of identified tasks}} \quad (\text{Eq. 2})$$

$$recall = \frac{\# \text{ of identified tasks} - \# \text{ of misidentified tasks}}{\# \text{ of identified tasks} - \# \text{ of misidentified tasks} + \# \text{ of missed tasks}} \quad (\text{Eq. 3})$$

7.2.2 Procedure for Evaluating Business Process Recovery Techniques

To measure precision and recall, we need to access fully documented business processes. However, in practice such documented processes do not exist and if the documentations exist they are usually incomplete. In our case study, we asked a research assistant (i.e., Mr. Kingchun (Derek) Foo) to manually inspect the source files and recovered business processes to verify the correctness (i.e., precision) and completeness (i.e., recall) of the recovered business processes. The inspector has experience developing business applications and did not participate in the development of our approach or our prototype tool.

The inspector studied the on-line documentation for the subject applications, and navigated through the different screens associated with each business process to better understand the overall structure of each process. The inspector used this acquired knowledge to determine the correctness of the recovered tasks.

To measure the precision of our approach for recovering a high-level process, the inspector executed the application and observed the screens in order to examine if the recovered tasks are business relevant or if they are misidentified tasks. To measure the recall of our approach, the inspector needed to identify tasks which are missing. To determine if there are any missing human tasks, the inspector ran through the screens of every process and compared the performed tasks with the tasks in the recovered processes.

7.2.3 Analyzing Results for Business Process Recovery

Table 7-2 presents the results of our case study. All of the recovered processes have precision and recall values which are above 99%. Table 7-2 summarizes the recovered high-level processes (HLP). For each application, we show the number of recovered high-level processes along with the number of human tasks recovered from screens, and the number of missed tasks as

Table 7-2. Evaluating the recovered high-level processes

Application	UI and controller				Precision	Recall
	#of HLP	#of human tasks	#of missed tasks	#of misidentified tasks		
SequoiaERP						
Accounting	24	43	0	0	1	1
Catalog	66	252	0	0	1	1
Content	47	92	0	0	1	1
Ecommerce	18	109	0	0	1	1
Facility	24	137	0	0	1	1
Manufacturing	25	59	0	0	1	1
Marketing	5	10	0	0	1	1
Order	25	297	0	0	1	1
Party	24	172	0	0	1	1
Workeffort	10	56	0	0	1	1
Opentaps						
Warehouse	25	488	0	0	1	1
Purchasing	26	192	1	0	1	0.995
Crmsfa	53	341	0	0	1	1
Financials	18	301	0	0	1	1
SMaCS						
SMaCS	99	61	0	0	1	1
HLP: High-Level Process						

well as misidentified tasks. Human tasks refer to tasks delivered by UI screens and require human interactions to the complete execution.

7.2.3.1 Analysis of Process Recovery Results for Studied Projects

Analysis of results for SequoiaERP: Table 7-2 presents the tasks recovered from the UI tier of the 10 applications of SequoiaERP. Our business process recovery approach achieves perfect precision and recall, since no missed or misidentified task is found.

Analysis of results for Opentaps: Table 7-2 summarizes the tasks recovered from the UI tier of the four applications of Opentaps. Different from SequoiaERP, one missed task in Opentaps is due to the fact that although multiple buttons in a screen may link to the same back-end component, the actual back-end components to be executed are determined at run-time based on the clicked UI button. Our static analysis cannot capture information generated at run-time. Our approach achieves an average precision of 100% and an average recall of 99.92%.

Analysis of results for SMaCS: as presented in Table 7-2, our approach achieves 100% precision and recall rates for the SMaCS project, because there is no missed or misidentified task found.

7.2.3.2 Analysis of the Use of UI Design Patterns for Identifying Tasks

Our approach uses UI design patterns to abstract the large number of UI widgets to a limited number of human tasks. We sought to closely examine the precision of our approach in abstracting UI widgets into human tasks. Table 7-3 shows the precision for the human tasks recovery in high-level processes using UI design patterns. The table also indicates the percentage of tasks that are identified using the UI design patterns compared with the total tasks recovered from UI screens. For example, in the accounting application our approach achieves a 100% precision in identifying human tasks. Out of the 43 identified tasks recovered from UI screens, 41 tasks are identified using UI design patterns. Other human tasks recovered from screens without using UI design patterns are mapped from some important UI widgets which cannot be grouped into UI design patterns. For example, a link may transit the current screen to an external component. Ignoring this link will make the recovered business process incomplete. Therefore, we map this link into a task. The precision of our approach for identifying human tasks using UI design patterns is 100% for all the studied applications. Our results show that UI design patterns result in simpler processes with business relevant tasks.

Table 7-3. Tasks recovered using UI design patterns

Application	Human task	
	Precision	Percentage
SequoiaERP		
Accounting	100%	41/43=95.35%
Content	100%	84/92=91.30%
Manufacturing	100%	50/59=84.75%
Marketing	100%	8/10=80.00%
Party	100%	67/172=38.95%
Facility	100%	87/137=63.50%
Catalog	100%	189/252=75.00%
Workeffort	100%	29/56=51.79%
Order	100%	154/297=51.85%
Ecommerce	100%	100/109=91.74%
Opentaps		
Warehouse	100%	261/488=53.48%
Purchasing	100%	143/192=74.48%
Crmsfa	100%	220/341=64.52%
Financials	100%	222/301=73.75%
SMaCS		
SMaCS	100%	61/61=100.00%

7.2.4 Threats to Validity and Limitations

We now discuss the different types of threats and limitations which may affect the validity of the results of our case study on recovering business processes.

External Validity: tackles the issues related to the generalization of the results. Among the 15 studied applications, 14 applications are based on the OFBIZ framework. While Opentaps has undergone several updates from the initial SequoiaERP version, both projects share many design and coding aspects. To address this potential bias, we chose to analyze SMaCS, a non-OFBIZ project. All the studied applications use data beans for accessing databases. We should in the future study the benefits of our approach using more projects to determine if our approach works well in other applications and domains.

In our case study, a research assistant, as a code inspector, evaluated the recovered processes. Our code inspector has experience in developing business applications and studied the on-line documentation from the applications. It is hard to find a professional business analyst to perform such time-consuming task. Business analysts usually have limited technical background. An analyst with limited programming knowledge would not be able to provide feedback on the effectiveness of our static tracing approach. In the future we plan to conduct a smaller evaluation using a business analyst. The analyst can provide more feedback on the meaningfulness of a small number of recovered processes and would help enhance our results so they are more acceptable in practice.

Internal Validity: is concerned with the issues related to the design of our case study. The manual inspection introduces bias since a single code inspector could make mistakes. We should have recruited additional code inspectors and evaluated the agreement between their analyses. Unfortunately, we were not able to recruit more code inspectors with sufficient knowledge about business processes and who can spend considerable time to manually inspect our results. In future, we plan to conduct our experiments involving more code inspectors to evaluate our business process recovery results. Based on the agreement between their analyses, we will get more accurate evaluations of our process recover techniques. Our use of static tracing has limitations as observed in the case study. We will explore the use of dynamic analysis in a future study.

7.3 Clone Detection

The following sections will discuss the case study applied to evaluate our clone detection techniques.

7.3.1 Measuring the Effectiveness of Task Clone Detection Techniques

Metric precision is used to evaluate the effectiveness of the results for identifying task clones. The precision metric (Eq. 4) evaluates the ability of our approach to correctly identify task clones. We cannot use recall criteria to evaluate the completeness of the detected task clones, since we are not able to get the complete set of all task clones existing in business processes.

$$precision = \frac{\# \text{ of identified task clones} - \# \text{ of misidentified tasks clones}}{\# \text{ of identified task clones}} \quad (\text{Eq. 4})$$

7.3.2 Procedure for Evaluating Clone Detection Techniques

For each studied application, we applied different thresholds to detect task clones from the UI tier and the business logic tier. We evaluated and compared the effectiveness of different thresholds to get ideal thresholds.

When we evaluated the effectiveness of our approach, we manually verified the detected task clones through comparing the corresponding code implementation and recognizing the refactoring opportunity. The task clones, which provide similar functionalities and have similar implementation containing code clones that can be resolved, are identified as correct task clones; the task clones, which are implemented by code fragments in similar format but deliver different functionalities, are recognized as misidentified task clones. We calculated the precision for each studied application. We also grouped similar task clone sequences into sub-processes and counted the number of identified sub-processes. A sub-process is composed of task clones and similar control flows and reflects the restructuring opportunity in business process level.

In the following sections we analyzed the experiment results got from the UI tier and the business logic tier of each studied application.

7.3.3 Analyzing Task Clone Results from UI Tiers

Table 7-4. Performance of our approach for detecting task clones from the UI tier using different thresholds

Similarity threshold value	# of identified task clones	#of correctly identified task clones	#of misidentified task clones	Precision
SequoiaERP				
40%	366	282	84	77.05%
50%	260	250	10	96.15%
60%	206	204	2	99.03%
Opentaps				
50%	249	229	20	91.97 %
60%	207	207	0	100.00%
70%	184	184	0	100.00%
SMaCS				
40%	22	20	2	90.91%
50%	20	20	0	100.00%
60%	15	15	0	100.00%

Table 7-5. Summary of detected task clones from the UI tier

Application	# of task clones	# of misidentified task clones	Total # of tasks	Precision	# of identified sub-processes	# of identified sub-process instances
SequoiaERP (Similarity threshold = 50%)						
Accounting	13	3	43	76.92%	0	0
Content	22	2	92	90.91%	0	0
Marketing	8	0	10	100.00%	0	0
Manufacturing	21	1	59	95.24%	0	0
Catalog	34	2	252	94.12%	0	0
Facility	40	1	137	97.50%	0	0
Order	52	1	297	98.08%	2	10
Party	18	0	172	100.00%	0	0
Workeffort	15	0	56	100.00%	0	0
Ecommerce	37	0	109	100.00%	0	0
Opentaps (Similarity threshold = 60%)						
Warehouse	53	0	488	100.00%	0	0
Purchasing	39	0	192	100.00%	2	4
Crmsfa	73	0	341	100.00%	3	6
Financials	42	0	301	100.00%	1	2
SMaCS (Similarity threshold = 50%)						
SMaCS	20	0	61	100.00%	0	0

Table 7-4 summarizes the performance of different thresholds to detect task clones located in the UI tiers. Table 7-5 presents the results for each studied application using the selected similarity threshold. All the applications are analyzed to detect task clones from the UI tier. For each application, Table 7-5 shows the number of detected task clones and the number of tasks recovered from UI screens. Table 7-5 also presents the number of misidentified task clones and identified sub-processes. The following sections will describe how to choose an ideal threshold value for each application and discuss the details of task clones detected from the UI tier of each studied application.

7.3.3.1 Analysis of Task Clones Located in the UI Tier of SequoiaERP

As summarized in Table 7-4, we chose three similarity thresholds (i.e., 40%, 50% and 60%) to evaluate the precision of our approach for detecting task clones in the UI tier of SequoiaERP. The higher similarity threshold achieves higher precision. The lower threshold values are able to identify more task clones. However, the number of misidentified task clones increases dramatically. With the similarity threshold value of 60%, the precision is 99.03% with only 2 misidentified task clones. Using the similarity threshold value of 40%, the precision is decreased to 77.05% with 84 misidentified task clones. The 84 misidentified task clones include the 2 misidentified task clones with similarity above than 60%, 8 misidentified task clones with similarity between 50% and 60%, and 74 misidentified task clones with similarity between 40% and 50%. These misidentified task clones are resulted from the code clones contributing to the common layout of tasks. For example, some misidentified task clones are mapped from input forms containing buttons defined by identical tags that make the implementing code blocks similar. However, the business data submitted by these forms and the back-end operations invoked by these forms are totally different. When we chose the threshold value, we aimed at keeping a balance between the ability to detect significant amounts of task clones and the ability

to filter out misidentified task clones. Therefore, a threshold value of 50% is the most ideal threshold since it achieves good precision in comparison.

Table 7-5 presents the results for each application in SequoiaERP that was studied using the similarity threshold of 50%. We manually verified each task clone by comparing the code fragments of the tasks identified as clones to check whether these code fragments are similar, deliver similar functionalities and suggest the refactoring opportunity.

The correctly identified task clones are implemented by identical or similar tags (e.g., table, form and link tags). For example, some HTML forms are copied and pasted into other applications or continue to be slightly modified to implement cloned tasks. These correctly identified task clones suggest the refactoring opportunity which involves resolving clones in the corresponding code fragments.

For the misidentified task clones (e.g., the two misidentified task clones from *Accounting* and *Manufacturing*) in the UI tier of SequoiaERP, the corresponding UI implementations have the similar UI layout structure, but work on different contents. The average precision for the task clones recovered from UI source code is 96.15%. We also identified 2 distinct sub-processes through grouping cloned tasks and identical tasks. Each sub-process has 5 instances which use the same control flows to coordinate cloned tasks or the same tasks.

7.3.3.2 Analysis of Task Clones Located in the UI Tier of Opentaps

As presented in Table 7-4, we tried different threshold values (i.e., 50%, 60% and 70%) to evaluate the precision of task clones detected from the UI tier of Opentaps. Both of threshold values 60% and 70% provide the 100% precision and the lower threshold value is able to find more task clones. Threshold 60% identifies 23 more correct task clones than threshold 70% does. Threshold 50% provides 22 more correct task clones than threshold 60% does, but also introduces 20 more misidentified task clones which have similarities between 50% and 60%. The misidentified task clones share similar layout but deliver different functionalities. For example,

two tasks respectively mapped from a table and a form are mistakenly recognized as task clones since the two widgets contain a lot of cells (e.g., TD tags) sharing similar layout attributes (e.g., width and alignment), but the functionalities delivered by the two tasks are different. The task mapped from tables allows users to browse information; the task mapped from forms supports users to edit and update business data. The 20 misidentified task clones decreases the precision from 100% to 91.97 %. Therefore, compared with values 50% and 70%, value 60% is more ideal.

Table 7-5 presents the details of clone results got from the UI tier of Opentaps using threshold 60% and amounts of detected task clones as well as identified sub-processes. No misidentified task clone is found from the results got by threshold 60%. The identified task clones are delivered by similar UI widgets implemented by cloned tags. Cloned overlapped code fragments also exist in some task clones and make the functionalities delivered by these tasks to be more similar. For example, one input form can participate in the implementation of multiple tasks. In Table 7-5, we got perfect precision for the results of Opentaps, and identified 6 different sub-processes each of which has 2 sub-process instances. The 2 instances of a sub-process deliver similar functional units by including groups of task clones.

7.3.3.3 Analysis of Task Clones Located in the UI Tier of SMaCS

As depicted in Table 7-4, we evaluated the effectiveness of detecting task clones with different threshold values (i.e., 40%, 50% and 60%) from the UI tier of SMaCS. Both of thresholds 50% and 60% can achieve perfect precision, and value 50% detects 5 more correct task clones than value 60% does. Therefore, value 50% is more effective than value 60%. Threshold 40% finds 2 more task clones than threshold 50% does, but these 2 task clones are misidentified. Therefore, value 50% has better performance in task clone detection than value 40%. The 2 misidentified task clones are caused by widgets that share similar layout structure, but work on different content.

Table 7-5 presents the details of task clones detected from the UI tier of SMaCS using threshold 50%. 20 task clones are found among all 61 tasks recovered from UI screens. No misidentified task clone is introduced by threshold 50%. Each task and its task clones, which are implemented with cloned tags (e.g., HTML and JSP tags), operate on similar business data and suggest the refactoring opportunity. There is no sub-process identified.

7.3.4 Analyzing Task Clone Results from Business Logic Tiers

Table 7-6 summarizes the performance of different threshold values to identify task clones from the business logic tier of each studied application. We tried different thresholds in order to find an ideal value that can recognize task clones with differences in the implementations and avoid introducing too many misidentified task clones.

Table 7-7 presents the task clone detection results from the business logic tier of each application with selected threshold values. Table 7-7 shows the details of clone detection results, such as the number of detected task clones, the number of misidentified task clones and precision for each application.

Table 7-6. Performance of our approach for detecting task clones from the business logic tier using different thresholds

Similarity threshold value	# of identified task clones	# of correctly identified task clones	# of misidentified task clones	Precision
SequoiaERP				
20%	69	65	4	94.20%
30%	69	65	4	94.20%
40%	64	60	4	93.75%
Opentaps				
20%	106	100	6	94.34%
30%	104	100	4	96.15%
40%	103	99	4	96.12%
SMaCS				
20%	26	24	2	92.31%
30%	24	24	0	100.00%
40%	24	24	0	100.00%

Table 7-7. Summary of detected task clones from the business logic tier

Application	# of task clones	# of misidentified task clones	Total # of tasks	Precision	# of identified-sub-processes	# of identified-sub-process instances
SequoiaERP (Similarity threshold = 30%)						
Accounting	19	0	189	100.00%	1	4
Content	14	0	132	100.00%	3	6
Marketing	0	0	0	N/A	0	0
Manufacturing	8	0	57	100.00%	1	2
Facility/Catalog	16	2	125	87.50%	6	12
Order	2	2	116	0.00%	0	0
Party	10	0	47	100.00%	3	6
Workeffort	0	0	5	N/A	0	0
Ecommerce	0	0	0	N/A	0	0
Opentaps (Similarity threshold = 30%)						
Warehouse	2	0	29	100.00%	0	0
Purchasing	0	0	39	N/A	0	0
Crmsfa	72	2	326	97.22%	12	24
Financials	30	2	170	93.33%	5	10
SMaCS (Similarity threshold = 30%)						
SMaCS	24	0	410	100.00%	1	2

The following sections will first describe how to choose the threshold value for each project and then discuss the details of task clones in business logic tier detected with the selected threshold values.

7.3.4.1 Analysis of Task Clones Located in the Business Logic Tier of SequoiaERP

Table 7-6 summarizes the results for detecting task clones from the business logics of SequoiaERP when three similarity threshold values (i.e., 20%, 30% and 40%) are selected. There is no difference of the results got by using threshold values 20% and 30% due to the same number of detected task clones and the same amount of misidentified task clones. The results for the threshold value of 40% give lower precision, since the correctly identified task clones are reduced. Therefore, the threshold value 20% or 30% is better than value 40%. In Table 7-6, thresholds 20%, 30% and 40% introduce the same 4 misidentified task clones since the 4 task clones have similarity above than 40%. These 4 misidentified task clones are mapped from

different method invocation statements in similar format and mistakenly identified as code clones, but different functionalities are delivered by these different method invocations.

Table 7-7 lists the results of detected task clones in the business logic tier of SequoiaERP, when the similarity threshold value of 30% is chosen. The detected task clones contain cloned code and cloned overlapped code. Cloned code fragments contribute to the implementations of multiple tasks and these cloned code fragments might be slightly modified, such as changing variable names. For example, two task clones' implementations can contain similar statement sequences using different variable names. The cloned overlapped code fragments can be shared by the implementations of multiple task clones and make the corresponding functionalities more similar. The restructuring opportunity is suggested by task clones. For example, task clones implemented by identical code blocks can be eliminated through procedure extraction [45].

As shown in Table 7-7, there are 4 misidentified task clones introduced in the business logic tier of SequoiaERP using threshold 30%. As have explained before, the misidentified task clones deliver distinct functionalities by invoking different methods. However, the methods have very similar structures and are identified as almost identical cloned statements. In the *Order* application, only two task clones are detected and misidentified due to above reason. As a result, the precision is 0. The applications *Facility* and *Catalog* share the same business logic tier, results for these two applications are shown in the same row. The average precision for detecting task clones from the business logic tier of SequoiaERP is 94.20%. From the business logic tier, we identified 14 different sub-processes which totally have 30 sub-process instances. The instances of the same sub-process use identical control flows to connect groups of task clones. .

7.3.4.2 Analysis of Task Clones Located in the Business Logic Tier of Opentaps

As presented in Table 7-6, we applied three different threshold values (i.e., 20%, 30% and 40%) on Opentaps to detect task clones from the business logic tier. Threshold 20% introduces 6 misidentified task clones and 4 of them are also introduced by thresholds 30% and 40%, since the

4 task clones have similarity above than 40%. The 4 misidentified task clones are resulted by the misidentified code clones (e.g., different method invocation statements in similar format) or the cloned utility statements which contribute to the similarity of code blocks mapped into tasks rather than the functionalities delivered by tasks. Another 2 misidentified task clones only introduced by threshold 20% are mapped from a set of misidentified code clones in similar format. For example, different assignment statements involving different types of variables on the left side and different method invocations on the right side are mistakenly recognized as near identical code clones due to the similar structure of the statements. The 2 misidentified task clones only introduced by threshold 20% make the result of threshold 20% has lower precision than the result of threshold 30%; the decreased number of correct task clones identified by threshold 40% leads the result of threshold 40% to have lower precision than the result of threshold 30%. Therefore, threshold value 30% has better performance than values 20% and 40% thanks to the precision and the number of detected task clones.

Table 7-7 presents the number of task clones detected from the business logic tier of each application from Opentaps under threshold value 30%. Similar as the situation in SequoiaERP, the detected task clones' implementations involve cloned overlapped code and cloned code that can be factored out by restructuring the source code.

We only found 4 misidentified task clones from applications *Crmsfa* and *Financials* as shown in Table 7-7. Two of the misidentified task clones are caused by different method invocation statements, which share similar format and are mistakenly recognized as almost identical code clones. However, different functionalities are delivered by the different method invocations and the corresponding tasks are not task clones. Another two misidentified task clones involve cloned utility statements in their implementations. These cloned utility statements are used to capture exceptions but still use business data, and therefore are grouped into corresponding tasks and make the implementing code fragments similar. However, the

functionalities delivered by the two misidentified task clones are still different, since there are few implementation code fragments similar except the cloned utility statements.

The overall precision for detecting task clones from the business logic tier of the four applications in Opentaps is 96.15%. We identified 17 different sub-processes each of which has 2 sub-process instances. The two instances belonging to the same sub-process coordinate task clones with the same control flows.

7.3.4.3 Analysis of Task Clones Located in the Business Logic Tier of SMaCS

As shown in Table 7-6, we tried different threshold values (i.e., 20%, 30% and 40%) to detect task clones from the business logic tier of SMaCS. The results got by value 20% contain two more task clones than the results obtained by value 30%, but these two task clones are misidentified since they are mapped from different code fragments (e.g., method invocations) in similar format and structure. This decreases the precision from 100% to 92.31%. Both of the results provided by value 30% and 40% contain the same amount of detected task clones and don't include any misidentified task clones. Therefore, the value 30% or 40% is better than value 20%.

Table 7-7 shows the details of the task clones detected from the business logic tier of SMaCS with threshold value 30%. The number of detected task clones and misidentified task clones are presented. Using threshold 30%, there is no misidentified task clone introduced. The detected task clones are contributed by both of cloned and cloned overlapped code fragments. These code clones, which are resulted by copy and paste actions applied in the task implementations, can be resolved through restructuring the source code. The restructuring process can result to resolve corresponding task clones.

We extracted 1 sub-process as a result of grouping cloned tasks and the same tasks. This sub-process has 2 instances that deliver similar functional units by connecting groups of task clones with the same control flows.

7.3.5 Summary of Task Clone Detection Techniques

The correctly identified task clones recovered from UI tiers are implemented by cloned tags (e.g., table, form and link tags) and deliver similar functionalities. Cloned overlapped codes can also appear in the implementation of task clones. For instance, one tag node can be involved with the implementation of multiple task clones to contribute to similar functionalities. The misidentified task clones located in UI tiers are mapped from UI widgets which share similar layout but work on different contents and provide different functionalities.

Correctly identified task clones in business logic tiers are implemented by cloned code fragments as well as cloned overlapped code fragments. Cloned code fragments can be identical or slightly modified. Cloned overlapped code fragments make the delivered functionalities more similar. Misidentified task clones in business logic tiers are caused by two reasons as follows:

- 1) Misidentified code clones are mapped into misidentified task clones. For example, misidentified task clones can be mapped from different method invocation statements sharing similar structure and recognized as code clones, but the functionalities invoked by these statements are different.
- 2) Utility code clones make the implementations of tasks similar but with different functionality. Specially, we use business data as heuristic [35][36] to identify business tasks. When utility functions take business data as input or return business data as output, such utility functions are misidentified as task implementations. Tasks implemented using cloned utility code may be identified as task clones due to the similar implementation, but the delivered the functionality may be different.

Chapter 8

Conclusion and Future work

We present an approach for automatically recovering business processes from user interfaces, and detecting task clones in business processes recovered from a business application. We design and develop the approach as well as integrate it into our BPE (Business Process Explorer) tool. The effectiveness of our approach is demonstrated through the case study using 15 business applications. In this chapter, we summarize the contributions of our thesis and discuss future work related to our research.

8.1 Thesis Contributions

Automatic techniques to recover business tasks and control flows from user interfaces: recovering business processes from UIs is a difficult task due to complex widget structures and arbitrary page transitions. We improve the work in [35][36] using UI design patterns to identify the starting point of business processes, capture tasks with proper granularity and identify control flows among tasks. Moreover, we implement the traceability between business tasks and UI code blocks through recording the position (e.g, line numbers and XPath expressions) of UI code blocks mapped into a task. With the traceability, software developers can quickly locate the code blocks corresponding to changed tasks and modify the code.

Automated strategies to detect task clones in business processes and refine business process definitions: detecting task clones in business processes is an important and challenging endeavor. Business analysts can use task clones to optimize and refactor business processes. However clone detection techniques operate and report results at the source code level. Our clone detection approach uses traditional source code clone detection techniques to detect clones in

CHAPTER 8. CONCLUSION AND FUTURE WORK

business applications and lifts the detected clones to the business process level. The results of the analysis are shown using business process entities (i.e., tasks and sub-processes) which business analysts are familiar with, instead of showing the results using methods and lines of code which software developers are more familiar with.

8.2 Limitations and Future Work

The work in this thesis has the following four major limitations. First, the evaluation of recovered business processes is conducted by a single source code inspector. The evaluation process are prone to be affected by the code inspector's bias and mistakes. Second, the business process recovery techniques applied on UIs rely on UI design patterns. However, if user interfaces of business applications do not involve with UI design patterns or just contain very few UI design patterns, our techniques may just be able to identify few business tasks from UIs. In this case, the recovered business processes may not effectively reflect the human tasks and control flows delivered by UIs. Third, our business process recovery techniques applied on UIs cannot identify parallel control flows, due the complexity that multiple threads of control can be executed autonomously and independently. Fourth, our current clone detection techniques do not analyze data flows which are important components of business process structures.

In the future, the work can be extended in following six directions:

Recruiting more code inspectors to evaluate the business process recovery results: we plan to recruit more code inspectors to verify the recovered business processes. Then, based on the agreement between their analyses we can get more fair evaluation of our business process recovery results.

Summarizing and recognizing more UI design patterns: we summarize the features of a few UI design patterns and detect these patterns so as to recover business processes from UIs.

CHAPTER 8. CONCLUSION AND FUTURE WORK

However, there are still many UI design patterns that we have not analyzed and might appear in the UI of business applications. A further study can be conducted to summarize and detect other UI design patterns, and extract control flows and tasks from these patterns.

Recovering business processes from other UIs except web-based UIs: our prototype tool supports to recover business processes from web-based UIs that are implemented with various languages such as HTML, XML and JSP. The web-based UI interacts with users through generating web pages and relies on web browsers. However, the UI of business applications can be implemented with other techniques (e.g., Java Swing [38]) and presented on computer monitors without relying on web browsers. In the future, we can implement parsers to analyze UIs which do not rely on web browsers.

Detecting parallel control flows from business applications: currently, we have not found an approach to identify parallel control flows from user interfaces due to the complexity of parallel control flows involving multiple threads of control. When we trace page navigations through static analysis, we are able to find the sequence, alternative and loop relations among pages. In the future, probably we can try to find UI design patterns which support parallel operations and extract parallel control flows from those patterns.

Clone detections in complete business process structures: our clone detection approach identifies clones in tasks and control flows. In the future, the clone detection work can be extended to the complete structure of business processes, such as including data flow components. Data flows play an important role in business processes. Given a syntactically correct business process, errors can occur during process execution due to incorrect data flows. A business task often delivers its functionality through processing business data. Processing identical business data, business tasks are prone to involve similar operations applied on these

CHAPTER 8. CONCLUSION AND FUTURE WORK

data. Therefore, the similarities of data flows of two tasks or processes can more or less reflect the similarity between the tasks or processes.

Refactoring clones detected in business processes: our clone detection approach focuses on detecting and denoting task clones as well as grouping task clones into sub-processes. The business process definitions are refined to explicitly denote the clone information which suggests the restructuring opportunities, but we haven't resolved the clones in business processes. The future work could resolve the detected clones (e.g., cloned tasks and identified sub-processes) to optimize business processes.

BIBLIOGRAPHY

Bibliography

- [1] G. Antoniol, R. Fiutem and L. Cristoforetti. Design Pattern Recovery in Object-Oriented Software. *Proceedings of International Workshop on Program Comprehension*, 1998, pp. 153-160.
- [2] G. Antoniol and Y.-G. Guéhéneuc. Feature Identification: A Novel Approach and a Case Study. *Proceedings of International Conference on Software Maintenance*, Sept. 2005, pp. 357-366.
- [3] B.S. Baker. On Finding Duplication and Near-Duplication in Large Software System. *Proceedings of Working Conference on Reverse Engineering*, July 1995, pp. 86-95.
- [4] H. A. Basit , S. Jarzabek. Detecting Higher-level Similarity Patterns in Programs. *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, Sep. 2005, pp. 156-165.
- [5] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna and L. Bier. Clone Detection Using Abstract Syntax Trees. *Proceedings of IEEE International Conference on Software Maintenance*, Nov. 1998, pp. 368-377.
- [6] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and Evaluation of Clone Detection Tools. *IEEE Transaction on Software Engineering*, vol. 33, no. 9, Sep. 2007, pp. 577-591.
- [7] T.J. Biggerstaff. Design Recovery for Maintenance and Reuse. *Computer*, vol. 22, no. 7, July 1989, pp. 36-49.

BIBLIOGRAPHY

- [8] C. Boldyreff and R. Kewish. Reverse engineering to achieve maintainable www sites. *Proceedings of Working Conference on Reverse Engineering*, Stuttgart, Germany, Oct. 2001, pp. 249-257.
- [9] Bauhaus Project. <http://www.bauhaus-stuttgart.de/bauhaus/index-english.html>.
- [10] Business Process Modeling Notation Specification. <http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf>.
- [11] F. Calefato, F. Lanubile and T. Mallardo. Function Clone Detection in Web Applications: A Semiautomated Approach. *Journal of Web Engineering*, vol. 3, no.1, 2004, pp. 3-21.
- [12] CCFinderX Quick Guide. <http://www.ccfinder.net/doc/quickguide-en.html>.
- [13] K. Chen and V. Rajlich. Case Study of Feature Location Using Dependence Graph. *Proceedings of Internal Workshop Program Comprehension*, June 2000, pp. 241-249.
- [14] E.J. Chikofsky and J.H. Cross II. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, vol. 7, no. 1, Aug. 2002, pp. 13-17.
- [15] CloneDR. <http://www.semdesigns.com/Products/Clone/>.
- [16] J.R. Cordy. Comprehending Reality - Practical Barriers to Industrial Adoption of Software Maintenance Automation. *Proceedings of IEEE 11th International Workshop on Program Comprehension*, Portland, Oregon, May 2003, pp. 196-206
- [17] J.R. Cordy, T. Dean and N. Synytskyy. Practical Language-Independent Detection of Near Miss Clones. *Proceedings of IBM Center for Advanced Studies Conference*, Toronto, Oct. 2004, pp. 29-40.

BIBLIOGRAPHY

- [18] G. Costagliola, A.D. Lucia, V. Deufemia, C. Gravino and M. Risi. Design Pattern Recovery by Visual Language Parsing. *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, 2005, pp. 102-111.
- [19] Designing Interfaces. <http://designinginterfaces.com>.
- [20] S. Ducasse, M. Rieger and S. Demeyer. A Language Independent Approach for Detecting Duplicated Code. *Proceedings of IEEE International Conference on Software Maintenance*, Aug. 1999, pp. 109-118.
- [21] A.B. Earls, S.M. Embury and N.H. Turner. A Method for the Manual Extraction of Business logics from Legacy Source Code. *BT Technology Journal*, 2002, vol. 20 no. 4, pp. 127-145.
- [22] Eclipse Platform. <http://www.eclipse.org/>.
- [23] Thomas Eisenbarth, Rainer Koschke and Daniel Simon. Locating Features in Source Code. *IEEE Transactions on Software Engineering*, vol. 29 no. 3, March 2003, pp.210-224.
- [24] W. Eixelsberger, M. Ogris, H.Gall, and B. Bellay. Software architecture recovery of a program family. *Proceedings of the International Conference on Software Engineering*, Japan, April 1998, pp. 508-511.
- [25] Expedia online booking system. <http://www.expedia.ca/>.
- [26] K.C. Foo, J. Guo, and Y. Zou. Verifying Business Processes Extracted from E-Commerce Systems Using Dynamic Analysis. *Proceedings of International Workshop on Program Comprehension through Dynamic Analysis*, Canada, 2007.
- [27] Freemarker Template Language,
http://fmpp.sourceforge.net/freemarker/dgui_template_overallstructure.html.

BIBLIOGRAPHY

- [28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley Publishing Company, Reading, MA, 1995.
- [29] A. Granlund, D. Lafreniere and D.A. Carr. A Pattern Supported Approach to the User Interface Design Process. *Proceedings of HCI International Conference on Human-Computer Interaction*, 2001.
- [30] J. Guo, K.C. Foo, L. Barbour, Y. Zou. A Business Process Explorer: Recovering and Visualizing E-Commerce Business Processes. *Proceedings of International Conference on Software Engineering*, Formal Research Demonstration Track, Leipzig, Germany, May 2008, pp. 871-874.
- [31] A.E. Hassan and R.C. Holt. Architecture Recovery of Web Applications. *Proceedings of the 24th International Conference on Software Engineering*, Florida, 2002, pp. 349-359.
- [32] D.M. Hilbert and D.F. Redmiles. Extracting usability information from user interface events. *ACM Computing Surveys (CSUR)*, v.32 n.4, pp.384-421, Dec. 2000
- [33] 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*; vol. 10, no. 1, pp. 3-35, 1998.
- [34] H. Huang. Business Rule Extraction from Legacy Code. *Proceedings of the 20th Conference on Computer Software and Applications*, 1996, pp. 162.
- [35] M. Hung and Y. Zou. Recovering Workflows from Multi Tiered E-commerce Systems. *Proceedings of International Conference on Program Comprehension*, Banff, July 2007, pp.198-207.

BIBLIOGRAPHY

- [36] M. Hung. A Framework for Extracting Hierarchical Workflows from E-commerce Systems. Mater thesis in Department of Electrical and Computer Engineering, Queen's University, Kingston, Canada, 2006.
- [37] IBM WebSphere Business Modeler.
<http://www306.ibm.com/software/integration/wbimodeler/>.
- [38] Java Swing. [http://en.wikipedia.org/wiki/Swing_\(Java\)](http://en.wikipedia.org/wiki/Swing_(Java)).
- [39] Z.M. Jiang, A.E. Hassan, and R.C. Holt. Visualizing Clone Cohesion and Coupling.
Proceedings of IEEE Asia Pacific Conference on Software Engineering, Bangalore, India, Dec. 2006, pp. 467-476.
- [40] J.H. Johnson. Substring Matching for Clone Detection and Change Tracking.
Proceedings of IEEE International Conference on Software Maintenance, Sept. 1994, pp. 120-126.
- [41] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A Multi-Linguistic Token-Based Code Clone Detection System for Large Scale Source Code. *IEEE Transaction on Software Engineering*, vol. 28 no. 7, July 2002, pp. 654-670.
- [42] R.K. Keller, R. Schauer, S. Robitaille and P. Page. Pattern-Based Reverse-Engineering of Design Components. *Proceedings of International Conference on Software Engineering*, 1999, pp. 226-235.
- [43] KLM online flights booking system. <http://www.klm.com/>.
- [44] R. Komondoor and S. Horwitz. Using Slicing to Identify Duplication in Source Code.
Proceedings of International Static Analysis Symposium, Paris, July 2001, pp. 40-56.

BIBLIOGRAPHY

- [45] R. Komondoor and S. Horwitz. Eliminating Duplication in Source Code via Procedure Extraction. Technical report, Computer Sciences Department University of Wisconsin-Madison, Madison, WI, USA, 2002.
- [46] K. Kontogiannis, R. Demori, E. Merlo, M. Galler, and M. Bernstein. Pattern Matching for Clone and Concept Detection. *Journal of Automated Software Engineering*, March 1996, pp. 77-108.
- [47] C. Krämer and L. Prechelt. Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software. *Proceedings of Working Conference on Reverse Engineering*, 1996, pp. 208-215
- [48] B. Laguë, E.M. Merlo, J. Mayrand and J. Hudepohl. Assessing the Benefits of Incorporating Function Clone Detection in a Development Process. *Proceedings of IEEE International Conference on Software Maintenance*, Oct. 1997, pp. 314-321.
- [49] F. Lanubile and T. Mallardo. Finding Function Clones in Web Applications. *Proceedings of European Conference on Software Maintenance and Reengineering*, March 2003, pp.379-386.
- [50] M.M. Lehman. Laws of Software Evolution Revisited. *Proceedings of the Fifth European Workshop*, 1996; Springer Verlag; 108-124.
- [51] G.A.D. Lucca, M. D. Penta, A.R. Fasolino and P. Granato. Clone Analysis in the Web Era: an Approach to Identify Cloned Web Pages. *Proceedings of the Seventh IEEE Workshop on Empirical Studies of Software Maintenance*, Florence, Italy, November 2001, pp.107-113.

BIBLIOGRAPHY

- [52] G.A.D. Lucca, M. D. Penta, and A. R. Fasolino. An Approach to Identify Duplicated Web Pages. *Proceedings of Annual International Computer Software and Applications Conference (COMPSAC)*, Oxford, England, Aug. 2002, pp. 481-486.
- [53] A. D. Lucia, G. Scanniello, and G. Tortora. Identifying clones in dynamic web sites using similarity thresholds. *Proceedings of International Conference on Enterprise Information Systems*, 2004, pp. 391-396.
- [54] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. *Proceedings of International Conference on Software Maintenance*, Nov. 1996, pp. 244-253.
- [55] A.M. Memon, I. Banerjee, and A. Nagarajan. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. *Proceedings of the Tenth Working Conference Reverse Engineering*, Nov. 2003, pp. 260-269.
- [56] M. Moore, S. Rugaber, and P. Seaver. Knowledge based User Interface Migration. *Proceedings of International Conference on Software Maintenance*, 1994, pp. 72-79.
- [57] Nokia Online Store. <http://www.nokia.co.uk/>
- [58] Opentaps Open Source ERP + CRM. <http://www.opentaps.org/>.
- [59] Patterns in Interaction Design, <http://www.welie.com/>.
- [60] D. Poshyvanyk, Y.-G. Gu  h  neuc, A. Marcus, G. Antoniol and V. Rajlich. Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. *IEEE Transaction on Software Engineering*, vol. 33 no. 6, July 2007, pp. 420-431

BIBLIOGRAPHY

- [61] D. C. Rajapakse and S. Jarzabek. An Investigation of Cloning in Web Applications. *Proceedings of International World Wide Web Conference*, Chiba, Japan, May 2005, pp. 924-925.
- [62] F. Ricca and P. Tonella. Analysis and Testing of Web Applications. *Proceedings of the International Conference on Software Engineering*, May 2001, pp. 25-34.
- [63] F. Ricca and P. Tonella. Using Clustering to Support the Migration from Static to Dynamic Web Pages. *Proceedings of International Workshop on Program Comprehension (IWPC)*, Portland, Oregon, USA, May 2003, pp. 207-216.
- [64] C.K. Roy and J.R. Cordy. NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. *Proceedings of International Conference on Program Comprehension (ICPC)* June 2008, pp. 172-181.
- [65] M. Salah, S. Mancordis, G. Antoniol, and M.D. Penta. Towards Employing Use-Cases and Dynamic Analysis to Comprehend Mozilla. *Proceedings of 21st International Conference on Software Maintenance*, Sept. 2005, pp. 639-642.
- [66] Sequoia Open Source ERP. <http://www.sequoiaerp.org/>
- [67] J. Shao and C.J. Pound. Extracting Business Rules from Information Systems, *BT Technology Journal*, 1999, vol. 17, no.4, pp. 179-186.
- [68] D. Sinnig. The Complicity of Patterns and Model-Based UI Development. Mater thesis in Department of Computer Science, University of Concordia, Montréal, Canada, 2004.
- [69] SMaCS. <http://sourceforge.net/projects/casualstaffhr/>.
- [70] H.M. Sneed and K. Erdos. Extracting Business Rules from Source Code. *Proceedings of the Fourth International Workshop on Program Comprehension*, 1996, pp. 240.

BIBLIOGRAPHY

- [71] H.M. Sneed and G. Jandrasics. Inverse Transformation of Software from Code to Specification. *Proceedings of the Conference on Software Maintenance*, 1988; 102-109.
- [72] H.M. Sneed. Software Renewal: A Case Study. *IEEE Software*, 1984, vol. 1, no. 3, pp. 56-63.
- [73] E. Stroulia, M. El-Ramly, P. Sorenson. From Legacy to Web through Interaction Modeling. *Proceedings of the 18th IEEE International Conference on Software Maintenance*, Oct., 2002, Montral, Canada, pp. 320-329
- [74] J. Tidwell. *Designing Interfaces*. O'Reilly Media, 2005.
- [75] Web Patterns,
http://harbinger.sims.berkeley.edu/ui_designpatterns/webpatterns2/webpatterns/home.php
- [76] A.J.M.M. Weijters and W.M.P. van der Aalst. Process Mining: Discovering Workflow Models from Event-Based Data.. *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence*, 2001, pages 283–290.
- [77] M. van Welie. Web Design Patterns. <http://www.welie.com/patterns/>.
- [78] M. van Welie, G. C. van der Veer, and A. Eliëns. Patterns as Tools for User Interface Design. *Proceedings of International Workshop on Tools for Working with Guidelines*, Biarritz, France, 2000, pp. 313-324.
- [79] Z. Xing and E. Stroulia. Understanding the Evolution and Co-Evolution of Classes in Object-Oriented Systems. *Journal of Software Engineering and Knowledge Engineering*, 2006; vol. 16, no. 1, pp. 23-52.
- [80] XML Path Language (XPath). <http://www.w3.org/TR/xpath/>.

BIBLIOGRAPHY

- [81] XML Process Definition Language. <http://xml.coverpages.org/XPDL20010522.pdf>.
- [82] Yahoo Design Pattern Library. <http://developer.yahoo.com/ypatterns/>.
- [83] Y. Zou, T.C. Lau, K. Kontogiannis, T. Tong and R. McKegney. Model-Driven Business Process Recovery. *Proceedings of the 11th Working Conference on Reverse Engineering*, 2004, pp. 224-233.
- [84] Y. Zou, Q. Zhang and X. Zhao. Improving the Usability of e-Commerce Applications Using Business Processes. *IEEE Transaction on Software Engineering*, 2007, vol. 33, no. 12, pp. 837-855.