Model-Driven Business Process Recovery

Ying Zou[1], Terence C. Lau[2], Kostas Kontogiannis[3], Tack Tong[2], and Ross McKegney[2]

Dept. of Electrical and Computer Engineering[1]
Queen's University
Kingston, ON, Canada
zouy@post.queensu.ca

IBM Canada Laboratory[2]
Toronto,ON, Canada
{lautc, tacktong,
rmckegne}@ca.ibm.com

Dept. of Electrical and Computer Engineering[3]
University of Waterloo
Waterloo, ON, Canada
kostas@swen.uwaterloo.ca

Abstract

A business process attempts to encapsulate the delivery of a sequence of tasks, typically starting from accepting a service request and ending at certain points, such as the completion of the service. In this paper, we propose a model-driven business process recovery framework that captures the essential functional features representing a business process. The framework utilizes static tracing techniques and a number of heuristics to map source code entities to high-level business process entities. A case study is preformed to recover IBM® WebSphere® Business Integration business processes from IBM WebSphere Commerce code. The experimental result demonstrates the effectiveness of the proposed framework.

1. Introduction

A business process can be defined as a set of interrelated tasks linked through a number of decision activities. Business processes have starting points and ending points, and they are repeatable. Moreover, business processes encapsulate the knowledge of operations and services provided by an organization. For example, the business process followed when a book is ordered may consists of a number of tasks such as checking the availability of the book, the need to restock the inventory, or the validity of the buyer's credit card.

Typically, a workflow represents a business process. It describes essential tasks, participants, business roles and resources required by the process. For the example of ordering a book, the workflow describes the participants (such as the buyer and the supplier), the tasks taken by each participant, and the order of the executions, along with the decisions for their executions.

Initially, the linkage between workflow entities and the underlying source code implementation can be established via the requirement specification and design documentation. However, the business application domain and the software implementation domain are subject to constant changes, and evolve independently. In the business domain, business processes are tailored to meet specific customers' requirements. Similarly, software development companies are continuously adding new functional features to their software products in order to keep their competitive edge. Thus, over time, the linkage between workflow entities and source code implementation drifts away from the initial documented linkage. It is a challenging task to maintain the consistency between the business workflow and the underlying source code implementation, especially when the functionality is deeply embedded in the existing source code and spread out in various physical locations.

To reflect the most up-to-date linkage between business tasks and their implementation in source code, we propose a model-driven business process recovery framework that extracts the *as implemented* business workflows from the source code and establishes associations between the business domain entities (such as tasks and decisions) and the implementation domain entities (such as methods and conditional constructs). We focus on the functional behaviors of business applications and the sequence of executions. Instead of tracing the requirements through a business application to recover a business process, we perform an automated analysis of the source code using heuristics to capture code fragments that implement business workflow entities. This analysis is performed through an abstract business process model that helps us associate relevant functional behaviors in the code with business workflow entities.

In this paper, our main focus is on the recovery of business processes from the source code of the information system implementing these processes. The

ultimate goal of our research is to synchronize changes that occur in business processes with ones in the information systems that implement these processes. This would help developers and businesses keep their business process documentation up-to-date. We envision that this approach would reduce the knowledge inter-dependency between the business domain and the software implementation domain. It would also assist businesses and developers perform impact analysis when changes in business processes require changes to the source code, and vice versa.

The remainder of the paper is organized as follows. Section 2 gives an overview of business workflows. Section 3 introduces e-commerce information systems, which are the focus of our research to recover business processes. Section 4 presents our model-driven business process recovery framework. Section 5 provides the heuristic rules for refining the recovery process. Section 6 describes the representation of the *as implemented* workflow. Section 7 discusses case studies that utilize the proposed approach. Section 8 introduces related work. Section 9 concludes the paper.

2. Business Workflows

A workflow describes a business process. It details various tasks involved in a business process and shows the decisions and rules that control the process. Furthermore, it describes the execution flow between these tasks. Table 1 gives the definitions of the workflow structural entities. The structural workflow entities are used to create the connected activity diagrams for representing workflows, such as the one in Figure 1. Business workflows are usually defined independently of the implementation domain and are mainly used by business people rather than technical experts. A business workflow contains tasks, data, loops, stops, decisions, choices and "goto" references, which can be visually represented using graphs, or stored in XML documents.

For example, Figure 1 illustrates an activity diagram for a business workflow, modeled in IBM WebSphere Business Integration Workbench. The depicted business process concerns the checking stale (line) items allocated from an order for an e-commerce vendor. The process' main goal is to periodically examine the status of stale items and process them appropriately based on a number of decisions. In Figure 1, the boxes represent tasks. The *"Time to Execute"* task starts the process asynchronously by a scheduler. The next task ("Find stale order line items") finds all stale items in all orders stored in the Order Management System. A set of business rules (defined elsewhere) specify the criteria for an item to be considered stale. For example, stale items could be allocated from an expected or back-ordered inventory. Each item is verified iteratively (indicated by the "goto" entity in the figure). Each item is checked to determine if it is still considered stale. Other decisions are checked based on the outcome of the source tasks. Finally, the output of the process is determined (either "Deallocate expected" or "Deallocate existing" task is processed).

Workflow Entities	Definitions
Tasks	A task is the lowest level of work that performs one logical step in a workflow activity diagram.
Decisions and Their Choices	A decision node describes the routing rules that a sequence of tasks must follow. A decision is followed by multiple or binary (Yes or No) choices.
Loops and Stops	A loop represents a repetition of a sequence of tasks. A stop node denotes a termination of the job.
Data	Data refers to the inputs/outputs for tasks.
Goto Reference	A goto reference represents the repetition of logic steps, and implements a loop.

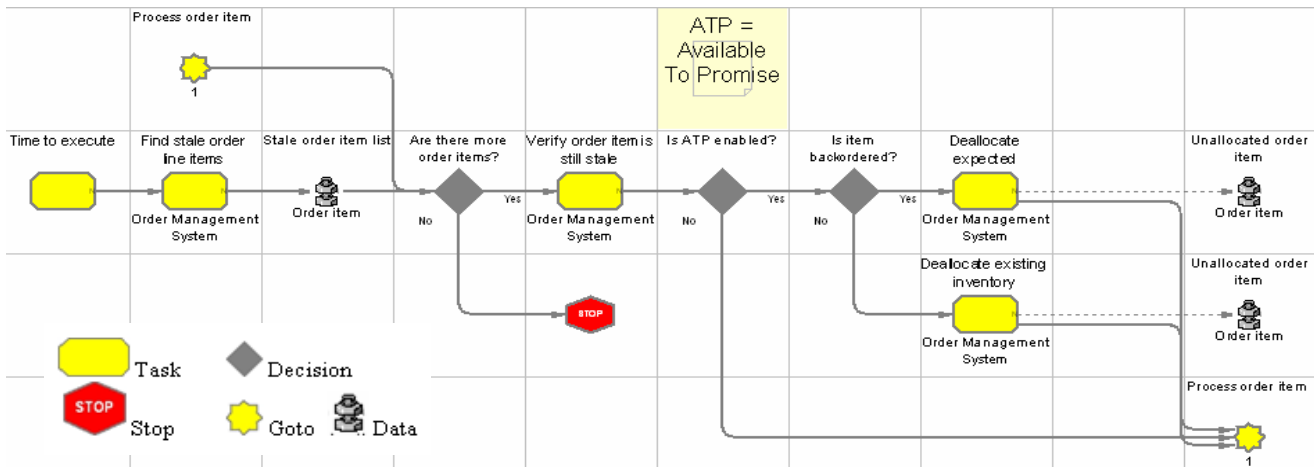Table 1: Workflow Structural Entities and Definitions [8]

Figure 1: An Example Business Workflow from IBM WebSphere Business Integration Workbench

Besides the workflow entities defined in Table 1, a workflow includes additional annotations that denote other details such as scheduling frequency (how often is the process executed, for example), and task annotations (such as resource requirements). In our current research, we are interested in gathering functional tasks (boxes in Figure 1), decisions (diamonds in Figure 1), and their connections to accomplish a business process from start to end. We are not able to recover scheduling information or task annotation from source code.

Our recovery framework may produce additional information that is not apparent in the business workflow. For example, the workflow shown in Figure 1 does not show a decision box to handle the case when an item is no longer stale; instead, that case is specified through free-form text annotation attached to the "verify order item is stale" task. Our recovery approach is likely to recover such a missing decision diamond that is used to completely implement such a task. The recovered information could be used to update the business process workflow with more accurate information detail or it can be noted aside while keeping the process workflow diagram unchanged.
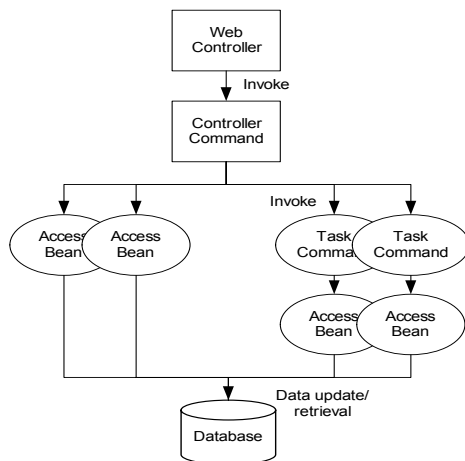

Figure 2: Controller-Centric Architecture for Web Applications

3. Business Information Systems

In our research, we focus on the information system that implements workflow for Web applications. In an implementation domain, an information system consists of software components, data variables, and execution conditions. These implementation domain entities each correspond to particular business process domain entities. For example, software components usually correspond to tasks; and execution conditionals correspond to decisions in the business workflow.

Information systems are deployed on scalable Web sites. A typical e-commerce Web application adopts the controller-centric architecture. This architecture approach utilizes Model-View- Controller (MVC) design pattern [5]. Figure 2 illustrates the fundamental structures in the architecture of a business information system [6]. In this architecture, a Web controller is built on top of the application to perform the central management for the client (Web browsers) requests. The Web controller is responsible for forwarding a client's requested Web pages to the appropriate controller command. In such an architecture, each Web page is not directly linked to another page. Instead, it is connected to its associated controller command. The controller command object is invoked in turn to complete a transaction. An example of a controller command is one that handles the ordering of books online.

The controller command serves a client's request by using access beans and task commands. The access beans are objects that retrieve/update business data in a relational database. For example, an access bean would retrieve details about the book being displayed such as the author's name and the date of publication. Task commands, known as software components, are usually designed using command design pattern [5]. For example, a task command may be responsible for ensuring that a book is available or if a book should be restocked. The task command uses access beans for data update/retrieval from a database.

In a well-designed information system, the code responsible to implement each task is encapsulated into objects, such as objects that extend task commands. However, in most cases, tasks are implemented as code blocks, which are scattered throughout the source code and are highly coupled with access beans. To recover a business process from source code, one of the challenges is to define fine-tuned criteria to identify source code, which corresponds to workflow entities.

4. A Framework for Model Driven Business Process Recovery

In this section, we present our model driven business process recovery approach, as depicted in Figure 3. The goal of our framework is to automatically recover the *as implemented* business workflow that abstracts source code entities to high-level business workflow entities. It also recovers the flow of control between source code entities. Once the *as implemented* workflow is obtained, we strive to synchronize the business workflow and the *as implemented* workflow.

The result of the *as implemented* workflow is defined in an abstract business process model that describes the commonality of entities in the business workflow domain and the information system domain. To produce data that conforms to this abstract business process model, the source code of the information system is analyzed automatically. The source code entities of interest are selected as potential candidates to represent business

workflow entities. A set of heuristics are used to reduce the entities that have been selected for analysis. These heuristics have been developed through consultation with developers and architects at IBM Canada. In addition, we recover the flow of control between the entities using static tracing techniques. We now describe each part of the framework.
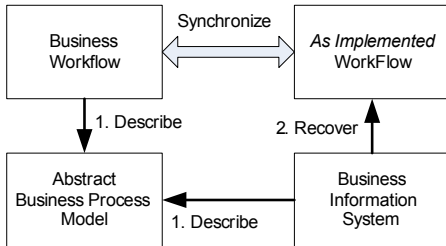

Figure 3: Framework for Model Driven Business Process Recovery

4.1. Abstract Business Process Model

A basic consideration for building an abstract process domain model is to bridge the gap between two domains: the business application domain (business workflow) and the implementation domain (business information system). By examining the basic structural entities in business workflows, as shown in Table 1, we identify the corresponding source code entities that are candidates for locating them in the implementation of information systems. The analysis of the business workflow entities and the implementation domain entities permits us to develop an abstract business process model for e-commerce applications.

Figure 4 illustrates the abstract business process model we developed. The *ControllerCommand* class refers to an object of type *Controller Command*. A *ControllerCommand* class is a likely candidate to contain a business process, corresponding to one business workflow. As depicted in the architecture of the information system (Figure 2), a controller command class implements one business process, which consists of a sequence of tasks along with conditions. Whereas a business workflow consists of abstract entities such as Tasks and Decisions; a Controller Command implementing such a workflow will contain code workflow entities, such as *Loop*, *TaskCommand*, *Decision* and *Choice*.

Specially, the *TaskCommand* class denotes business logic pieces that are implemented as classes. The *Task* class refers to code fragments that implement pieces of business logic. Essentially, a *Task* object gathers several related method invocations and access beans that access a database.

The *Decision* class describes evaluation expressions occurring in *if*, *while,* and *for* statements. The

Decision class corresponds to the decision boxes used in business process workflows. The *Choice* class specifies conditions that lead to *Yes* or *No* branches. Moreover, the *Yes/No* branches may contain, in turn, a sequence of code workflow entities.

The *Loop* class represents iterations of processing steps that are encapsulated in *for*, *while*, and *do* statements. Normally, such a statement starts with an evaluation expression that specifies the conditions for the termination of an iteration. Therefore, the *Loop* class contains a *Choice* class that describes the termination of the iteration.
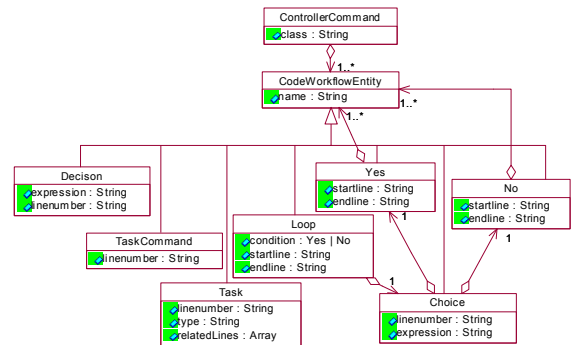

Figure 4: Abstract Business Process Model

In addition, the attributes, such as *linenumber*, *startline,* and *endline*, refer to the code range or position of a particular entity in source code. To facilitate the workflow synchronization, we link the *as implemented* workflow with source code, using the line numbers where each source code entity appears in the source code. Once changes are detected in the business workflow, we can locate the affected code workflow entities in the *as implemented* workflow via the associations between the two workflows. Furthermore, we can determine the affected code areas in the information system, using the line numbers specified in the *as implemented* workflow. Similarly, the changes in source code can also be propagated into the business workflow.

4.2. Workflow Recovery Process

Conceptually, the business workflow consists of a trace record of the execution of an application. Unfortunately, generating a complete trace record of an application would produce a workflow graph that is likely to contain a large number of irrelevant code entities that do not map to business workflow entities. To overcome this problem, we use a set of heuristics to prune the number of code entities included in the trace records. These heuristics are derived based on our experience in developing e-commerce applications and are dependent on the type of Web application technologies used. In our research, the e-commerce application is implemented in Java™ code

along with design patterns, as explained in Section 3. Our recovery approach is fully automated and it is integrated in the development environment, thus enabling easy and immediate access for developers to monitor changes to their implemented workflow. The process is described in the following subsections.

4.2.1. *Trace Record Generation*

Generally, a trace records a sequence of operations executed in an application. Dynamic traces are performed by executing the application upon providing different input values. Static traces, on the other hand, track the potential sequence of operations without executing the application. Static traces are independent of the input and do not require the setup and execution of the application.

In our research, we attempt to analyze enterprise level e-commerce applications. Static traces are more convenient as they do not require highly skilled personnel and computation resources to install and operate such large-scale applications. Moreover, static traces give a complete record of the possible execution paths of an application. In short, the static tracing starts from the entry of the main method of the controller object in a Web application, and ends at the exit of that main method. The extraction process visits every statement in the method body in sequence, and recursively follow the call path of the method. We perform the static tracing using an approach similar to [3, 7].

4.2.2. *Trace Record Refinement*

Information systems are workflow applications that are written in Java and used in e-commerce environments. A trace record for such applications would contain a large sequence of operations that are too low level to have equivalent representation in the business workflow; therefore, we must prune the paths of static traces. The refinement process filters excessive code-oriented information, and merges small fine-grained code fragments implementing business logic into higher-level coarse-grained business workflow entities. The process also generates a pseudo name to denote the aggregate business logic.

To perform the pruning, we first need to parse the source code of the Web application, and create a type repository. The type repository stores the names of all Java classes defined in the application. We adopt the parser built in Eclipse [9] for parsing the source code and generating an Abstract Syntax Tree (AST) of the source code. Eclipse is used for a Java programming development environment and an open source tool integration platform. In particular, we utilize the Java Document Object Model (JDOM) provided by Eclipse

Development Tools (JDT) to analyze the structures of Java programs. An Abstract Syntax Tree (AST) of Java source code can be accessed via the JDT API. We traverse the AST to analyze source code as a tree of nodes, where each node represents a part of the source code (for example, CaseStatement, DoStatement, IfStatement, Literal, TryStatement).

As depicted in Figure 2, e-commerce applications adopt controller-centric architecture where a controller command object provides a single entry point for intercepting HTTP requests coming from end users. The controller command manages data and control flows to accomplish a single business process. It makes use of task command objects and access beans. In this context, our tracing focuses on the source code of the controller commands, and ignores the code inside task commands and access beans. In addition, we apply heuristic rules to map source code entities to business workflow entities. As an output of the static tracing, the *as implemented* workflow is extracted from the source code. It is stored in an XML format, which conforms to the abstract business process domain model, as depicted in Figure 4.

4.3. *As implemented* Workflow and Business Workflow Synchronization

Once the *as implemented* workflow is generated, we aim to establish the linkage between the business workflow and the *as implemented* workflow. In this context, it is important that the two workflows have similar structures. In this way, entities in the *as implemented* workflow can be easily recognized and compared with the entities in the business workflow. Therefore, our goal is to automatically convert the XML represented *as implemented* workflow into a graph that is also used to represent business workflows. A software analyst is responsible for comparing entities in both workflows and mapping them to each other. In addition to the domain knowledge required for establishing the mapping, this process is also assisted by the automated naming technique that is used during the static tracing process. For example, a business workflow task named "verify order item is stale" is likely to be mapped to a recovered task named "verifyOrder", which in turn is assigned based on the name of a Java method.

5. Heuristics for Trace Record Refinement and Workflow Mappings

As described in the previous section, our recovery approach makes use of a number of heuristics to reduce the complexity of the generated traces and to map the traces to business process entities. In this section, we present these heuristics and explain our intuitions behind using them.

5.1. Workflow Mapping Heuristics

To extract the *as implemented* workflow from the source code, we identify a set of mapping rules that associate the entities in the business workflow with the code entities in the source code. As listed in Table 2, column 1 specifies the entities used in a business workflow. Column 2 denotes the related code entities. For example, a task command object in the source code represents the task entity in a business workflow. Moreover, the mapping from workflow entities to code entities is a one-to-many relation. For instance, the `Choice` workflow entity is related to `if/switch` statements. Such source code entities are used to compose the *as implemented* workflow. To gather the source code entities, we developed a feature-based technique that focuses on examining method invocations, object declarations, and database access statements to capture all possible candidates for the *as implemented* workflow entities.

Workflow Entities	Code Entitites in Source Code
Task	Task commands, code fragments, access beans
Data	*Vector* objects, *Enumeration* objects, *List* objects, access beans
Decision	Evaluation expression in *while*, *for* and *do* statements
Choice	*then*/*else* branches in *if* statement, Switch statement,
Loop	Loop bodies in *while*, *for* and *do* statements
Stop	*return* statement
Goto reference	*goto* statement, and end of loop body

Table 2: Mapping between Work Entities and Source Code Features

5.2. Trace Record Refinement Heuristics

In most cases, workflow entities can be located in source code by strong evidence of domain knowledge. For example, *Decision* entities are cognitively relevant to the evaluation expressions in *while*, *for,* or *do* statements. Ideally, the *Task* workflow entity, representing a business processing step, is explicitly implemented as *Task Command* objects. However, in the real world of development, the *Task* workflow entity can also be implemented as code fragments that are not developed as objects. In this context, to successfully recover a complete business workflow from applications, it is critical to define fine-tuned criteria that facilitate effectively locating possible *task* workflow entities (i.e., pieces of business logic). Essentially, the basic principle for detecting task workflow entities in source code is that the code fragments implement business services (e.g.,

ordering of books) and comply with business rules. Normally, business rules are associated with database access and data validation. An example of a business rule is to verify whether the item is in a shopping cart.

In our study, we focus on the IBM e-commerce systems with the architecture shown in Figure 2. To locate source code implementing workflow entities from the IBM e-commerce code, we developed a set of heuristic rules for filtering irrelevant code entities based on the characteristics of the code. To this end, during the traversal of the AST generated by the static trace technique, we focus on gathering potential candidates for task workflow entities. In the rest of this subsection, we present the heuristic rules that are used to determine whether code features can be extracted as task workflow entities from IBM e-commerce applications.

```
1. OrderAccessBean abOrderHelper =
2.      new OrderAccessBean();
3. Vector vItems = abOrderHelper.findValidItems(storeId);
4. Enumeration enumItems = vItems.elements();
5.
6. getContext();
7.
8. try{
9.      Transaction.commit();
10.     }
11. catch (javax.transaction.RollbackException ex){
12.     throw new Exception(......);
13. }
```

Figure 5: Sample Controller Command Code

Rule 1 — Utility code: Utility code, such as utility objects or methods, provides internal services that facilitate the completion of a business process. Such objects are not part of a business process and instead act as helpers; therefore, they should not be extracted. For example, a utility object performs transaction initialization, commitment, rollback, logging, or tracing. Generally, utility methods are designed as public static methods that can be invoked globally inside a Web application. As shown in line 9 in Figure 5, the `Transaction` class offers a global method to commit a transaction. Its method `commit()` is invoked directly via its containing class without an object declaration in the body of its calling method. By these characteristics, we can detect utility code entities. Furthermore, a catalog of utility classes can be established by consulting developers.

Rule 2 — Java type objects and their methods: A Java type class, such as String, Enumeration, and Vector, provide primitive building blocks to construct applications. Code fragments based on these basic types are not considered to represent task workflow entities, and therefore are not considered in the recovery of the business process. For example, as illustrated in line 4 in

Figure 5, the `elements()` method in `Enumeration` is commonly used in Java applications. Therefore, the static tracing ignores that method call.

Rule 3 —— Exception objects and their methods: A user-defined exception class extends an abstract exception class provided by the Java API. Typically, exception objects are responsible for error handling without encapsulating business logic. An exception object can be easily identified in catch statements, as illustrated in line 11 and 12 in Figure 5, or in throw statements. In some cases, exception may be used to report error cases in the business process handling; we currently do not handle such cases.

Rule 4 —— Access bean objects and their methods: An access bean object is an object wrapper that encapsulates the operations for data retrieval and updates. It mainly contains a set of trivial methods (i.e., a set of getters and setters) that take parameters, compose database access statements, and populate database results. Moreover, access bean objects perform a certain set of business rules, such as data item validation, approval, and removal. As illustrated in Figure 5, `abOrderHelper` is an access bean object of the type `OrderAccessBean`. The method `findValidItems()` finds items from the database, and also employs business rules to validate the returned items. This type of method is a good candidate for a task workflow entity implementation. Furthermore, we use the name of this method as the name of the recovered *task* workflow entity. Access bean objects can be distinguished by the class inheritance structure. A specialized access bean extends an abstract access bean. Using this rule, we collect non-trivial methods of access beans appearing in controller command objects. These non-trivial methods are considered as candidates for *task* workflow entities.

Rule 5 —— Task command objects and their methods: A task command class adopts the command design pattern, in which a command interface extends an abstract command interface and provides a concrete implementation of its action. We examine the class inheritance relations and interface implementation declarations in the AST representation of the source code. If a task command class is an extension to an abstract task command, we collect its task command objects as *task* workflow entities. In this context, the name given to the task workflow entities in the recovered workflow is the name of the task command class.

Rule 6 —— Methods inside controller commands: A controller command class can define its own methods. The methods can be either trivial methods (i.e., a set of setter and getter methods) or user-defined methods. To discover the *task* workflow entities, we ignore the trivial methods, but further analyze the statements inside the body of user-defined methods.

Rule 7 —— Other user-defined class objects and their methods: As previously mentioned, one basic principle to determine whether a non-trivial method in user-defined classes contains any business logic is to check whether the method involves database accesses and employs business rules to handle data items. In this case, the *task* workflow entity detection involves examining the database usage and population. Furthermore, we check whether a business rule is applied to validate the data, for example, examining whether the data is used in decision expressions. In this context, the name of the detected *task* workflow entities is taken from the name of the user-defined methods.

```
<?xml version="1.0" ?>
- <ControllerCommands>
 - <ControllerCommand class="ReleaseExpiredAllocationsCmdImpl">
   - <Task linenumber="152" name="startUse" type="source">
     - <RelatedLines>
         <Line number="152" />
         <Line number="261" />
       </RelatedLines>
     </Task>
   + <Task linenumber="163" name="findStaleOrderItems" type="source">
     <Decision expression="hasMoreElements" linenumber="177" />
   - <Loop condition="yes" endline="232" startline="177">
     + <Task linenumber="186" name="verifyStaleOrderItems" type="source">
     - <Choice expression="abOrderJDBCHelper.verifyStaleOrderItems(storeId,orderitemsId)" linenumber="186">
       - <Yes endline="225" startline="186">
         + <Task linenumber="195" name="IsUsingATP" type="source">
         - <Choice expression="bATPEnabled" linenumber="196">
           - <Yes endline="214" startline="197">
             - <Choice expression="abOrderitem.getInventoryStatus().toUpperCase().equals(ALLC)" linenumber="200">
               - <Yes endline="205" startline="200">
                 + <TaskCommand linenumber="201" name="DeallocateExistingInventoryCmd">
                 </Yes>
               - <No endline="210" startline="206">
                 - <Choice expression="abOrderitem.getInventoryStatus().toUpperCase().equals(BO)" linenumber="207">
                   - <Yes endline="212" startline="207">
                     + <TaskCommand linenumber="208" name="DeallocateExpectedInventoryCmd">
                     </Yes>
                   </Choice>
                 </No>
               </Choice>
             </Yes>
           </Choice>
         </Yes>
       </Choice>
     </Loop>
 </ControllerCommand>
</ControllerCommands>
```

Figure 6: Example of an *As Implemented* Workflow in XML Representation

6. *As Implemented* Workflow Representation

In Section 4, we discussed our workflow recovery process that traces the source code, and identifies workflow entities and their execution paths, along with the conditions of their executions. Meanwhile, the result of the *as implemented* workflow is represented in an XML format that conforms to the abstract business process model specified in Figure 4.

Figure 6 demonstrates a sample of an *as implemented* workflow. This workflow is extracted from the controller command that implements the process "release expired

allocation" that was depicted in Figure 1. Task workflow entities are described by either <task> or <taskcommand> in the XML document. The <task> tags refer to the *task* workflow entities that are extracted from method invocations. The name attribute of each <task> tag refers to the name of the invocated method. The <taskcommand> tags represent the *task* workflow entities that are explicitly encapsulated in task command objects. The <choice> tags correspond to `if` statements. The expression attribute of a <choice> tag is generated from the evaluation expression of the related `if` statement. In addition, the <RelatedLines> contains the line numbers of code fragments that implement workflow entities.

Moreover, the *as implemented* workflow in Figure 7 is depicted in a graph by interpreting the XML document in Figure 6. We follow the same legends used by IBM WebSphere Business Integration Workbench, shown in Figure 1.
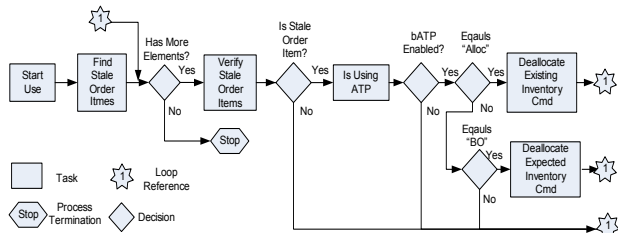

Figure 7: *As Implemented* Workflow for the Process of Releasing Expired Allocations

7. Case Study

To demonstrate the effectiveness of the framework advocated in this paper, we developed a re-engineering tool that automatically extracts the *as implemented* workflow from e-commerce applications. This tool is fully integrated in Eclipse IDE as an Eclipse plug-in. We tested the tool on workflow applications, which are IBM WebSphere Commerce applications. The detailed case studies and experimental results are presented in this section.

7.1. IBM WebSphere Commerce

IBM WebSphere Commerce is a family of products for building e-commerce Web sites and applications. The product line provides B2B and B2C market models for creating on-line stores, catalogs, and campaigns. The business process workflows are defined using IBM WebSphere Business Integration Workbench, independent from the IBM WebSphere Commerce platform. IBM WebSphere Business Integration Workbench is used to create workflows, and provides a collection of default business workflows for e-commerce market models. Typically, business workflows are customized to meet the requirements of different

merchants. The WebSphere commerce code will change from version to version. In this case, we aim to synchronize the IBM WebSphere Business Integration business workflows with the workflows implemented in the WebSphere Commerce code. By applying the proposed model-driven business process recovery approach, we successfully recover the workflow encoded in the WebSphere Commerce applications. The extracted workflow captures the essential workflow entities and their interactions. The linkage of workflow entities between IBM WebSphere Business Integration business workflows and the *as implemented* workflows is currently established manually.

IBM WebSphere Business Integration Workflow Entities	*As Implemented* Workflow Entities
Time to execute	StartUse
Find stale order line items	FindStaleOrderItems
Verify order item is still stale	VerifyStaleOrderItems
Deallocate expected	DeallocateExpectedInventoryCmd
Deallocate existing inventory	DeallocateExistingInventoryCmd

Table 3: Workflow Associations for Releasing Expired Allocations

7.2. Experiment Results

In this section, we evaluate the usefulness and the applicability of our proposed approach. We selected 86 applications from IBM WebSphere Commerce code for our experiment. We present two sample processes including releasing expired allocations (illustrated in Figure 1 and 7) and processing back orders (illustrated in Figure 8). Moreover, the "allocate inventory" task in the back orders process contains a subprocess called allocating inventory, which is expanded by a sequence of tasks, as shown in Figure 8. To verify the correctness and completeness of the recovered *as implemented* workflow, we compare it with the exiting business workflows specified by IBM WebSphere Business Integration Workbench. Table 3 and Figure 8 illustrate the linkages for these two processes. The linkages are established between the extracted workflow entities and the tasks of the business workflows. By comparing the entities between the business workflow and the *as implemented* workflow, we conclude that the *as implemented* workflows from these two processes are matched with the IBM WebSphere Business Integration workflows. Other recovered *as implemented* workflows are verified by IBM developers. The tool can be integrated in the IBM WebSphere Application Development environment, an Eclipse-based IDE, to extract *as implemented* workflows from source code for keeping the documentation up-to-date.
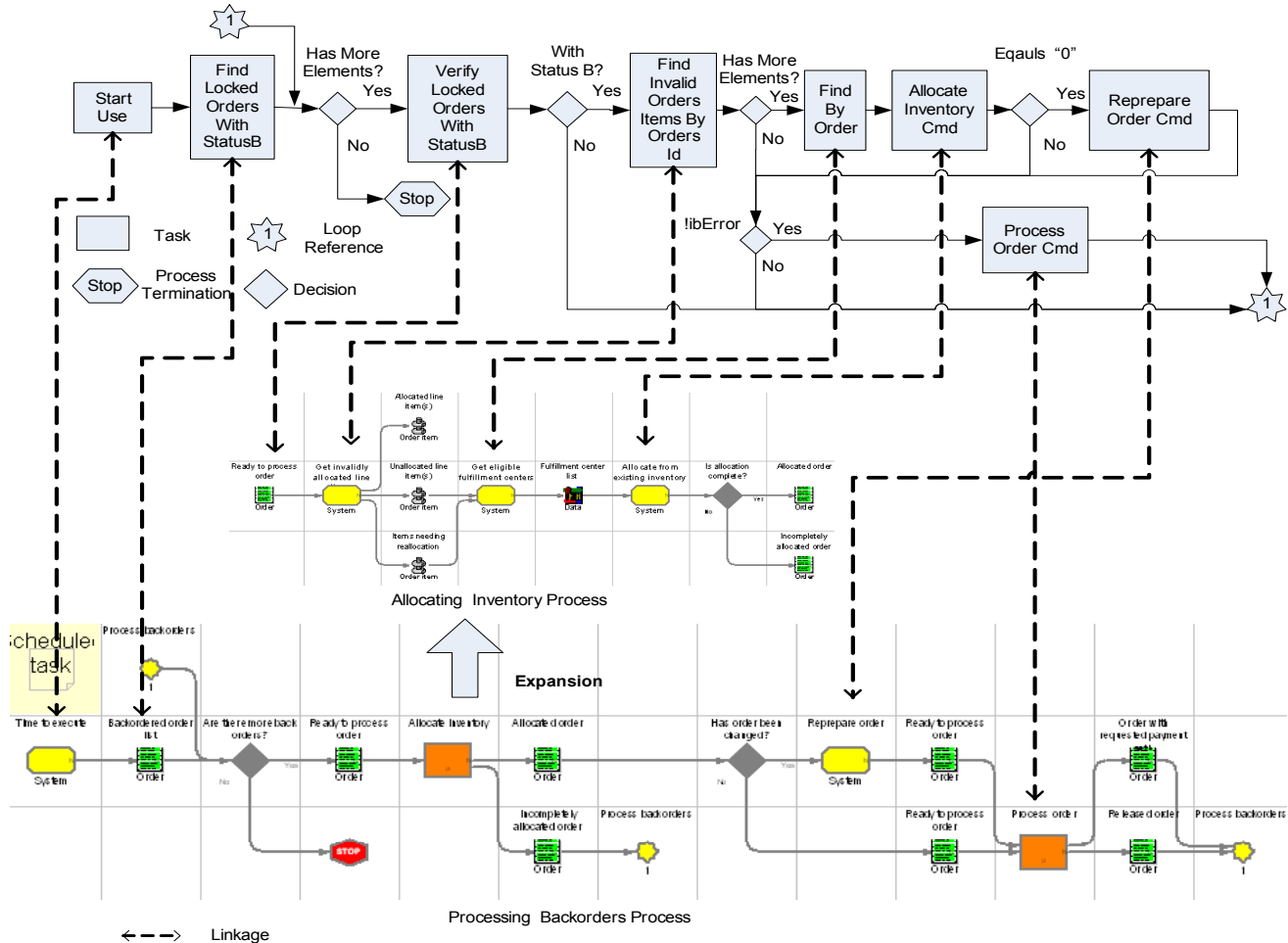
Figure 8: The Comparison between the *As Implemented* Workflow and the Business Workflow for Processing Backorders

Discussion: The structure of the *as implemented* workflows is relevant to the functional behaviors of the application. It contains programming-oriented decision branches, and uses ad hoc naming conventions adopted by the developers. It may be difficult for business analysts to understand the structure. In this context, we cannot replace the business workflows with the *as implemented* workflow. However, the *as implemented* workflow can be used to inspect the correctness of business workflows, and refine the business workflows. Moreover, the *as implemented* workflow can serve as a template to guide the creation of a business workflow.

The *as implemented* workflow contains the essential *task* workflow entities in the code, and filters out unnecessary information. This facilitates establishing up-to-date associations between the entities of business workflows and the ones in source code that implement business workflows. The *as implemented* workflow and the associations can indicate whether an information system fully supports the corresponding business workflow. In some cases, a system might only partially implement a workflow.

Moreover, the associations can also indicate the need for refactoring source code. For example, one of the premises in designing a workflow application is to implement each *task* workflow entity as a task command object. In the case that a task workflow entity is implemented as a code fragment, this indicates that such a code fragment should be refactored and implemented as a separate task command object.

8. Related Work

Typically, a business process model is concerned with the functional behaviors of a software system. It can be represented by UML to describe essential events, resources, input/output and procedures that govern a sequence of business activities. A great deal of effort has been devoted to bridge this significant gap. Koehler *et al.* define an IT flow model that provides a high-level abstraction to the detailed source code [3]. A pattern-based mapping connects business models with IT flow models. Consequently, process requirements are translated into logical formulas, which can be verified

using model checking techniques. However, the business process model cannot be traced from the source code. Di Lucca *et al.* provide heuristic criteria to restructure business-level UML diagrams from source code [4]. In this case, the business process model needs to be recovered manually by human experts from the UML diagrams.

Eisenbarth *et al.* describe an automatic technique to extract static traces for individual stack and heap objects using call graphs [6]. Tonella *et al.* present a technique for the automatic extraction of UML interaction diagrams from C++ code [7]. These techniques focus on tracing source code to detect method invocations between objects. Our static trace technique presented in this paper focuses on detecting business logic by traversing the AST and walking through statements. Liu *et al.* present a semiotic approach to recover requirements from legacy systems by analyzing and modeling behaviors [2]. The approach relies on executing legacy systems, and captures the input/output data to the systems. Our approach is solely based on analyzing the source code of information systems.

9. Conclusion

In this paper, we presented a model-driven re-engineering framework for recovering business processes from source code, and establishing the associations between business workflow and the *as implemented* workflow. We addressed the issues, including the representation of an abstract business domain model, the detection of business workflow entities in the code, and the extraction of the *as implemented* workflow. We believe that this model-driven business process recovery framework facilitates the change management of business workflows and information systems during the evolution and maintenance process. A plug-in tool was developed to extract *as implemented* workflows from Java-based Web applications.

This is an ongoing collaborative research project of Queen's University at Kingston, University of Waterloo, and IBM Centers for Advanced Studies. The future extensions to this research focus on refining the criteria on the detection of business logic in more general source code, such as open source applications without using controller-centric architecture. Furthermore, we plan to investigate the potential for automatically establishing the associations between business workflows and the *as implemented* workflows.

Trademarks

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

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

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

References

[1]   Thomas Eisenbarth, Rainer Koschke, and Gunther Vogel, "Static Trace Extraction", in the proceedings of 9th Working Conference on Reverse Engineering, 2002.

[2]   Kecheng Liu, Albert Alderson, Zubair Qureshi, "Requirements Recovery from Legacy Systems by Analysing and Modelling Behaviour", in the proceedings of International Conference on Software Maintenance, 1999.

[3]   J. Koehler *et al.*, "From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods", in the proceedings of the International Enterprise Distributed Object Computing Conference, 2002.

[4]   G. Di Lucca *et al.*, "Abstracting Business Level UML Diagrams from Web Applications", in the proceedings of the International Workshop on Web Site Evolution, 2003.

[5]   Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, "Design Patterns", Addison-Wesley, Pub. Co. 1995.

[6]   Bill Moore, Peter Gothager, Michael Mattinson, Chiara Montecchio, Narayan Prasad, Carla Sofia Jesus Ribeiro, and Thomas Tolborg, "WebSphere Commerce V5.4 Developer's Handbooks", IBM Red Books Series.

[7]   Paolo Tonella and Alessandra Potrich, "Reverse Engineering of the Interaction Diagrams from C++ code, in the Proceedings of International Conference on Software Maintenance, 2003.

[8]   User Guide, IBM WebSphere Business Integration Workbench.

[9]   http://www.eclipse.org.