Mining Reuse Processes

Talita L. Gomes[1], Toacy C. Oliveira[1], Don Cowan[2], Paulo Alencar[2]

[1]PESC – COPPE – Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brasil

[2]David Cheriton School of Computer Science, University of Waterloo,
Waterloo – ON – Canada

{talitalopes,toacy}@cos.ufrj.br
dcowan@csg.uwaterloo.ca, palencar@cs.waterloo.ca

Abstract. Object-oriented frameworks are widely used for improving software reuse. However, learning how to use these frameworks may be a difficult task. Developers need to know which classes and methods to reuse (reuse actions) and in which order they should be implemented. In this context, this paper describes how to support the software reuse task by mining reuse actions from existing framework applications. The reuse actions are mined from a code repository and they are used as input to discover the underlying instantiation process, using process discovery algorithms from process mining.

Keywords: Software Engineering, Software reuse, Object oriented frameworks, instantiation process, process mining, mining software repositories, Reuse description language, RDL

1 Introduction

Object-oriented frameworks are developed with the goal of reducing the time and effort needed for the development of software applications. To achieve this goal, frameworks capture the design and implementation of a certain domain from which a wide range of applications can be built [1].

Framework applications are implemented by augmenting the flexible parts of the framework. To do so, developers must be aware of what classes and methods they should enhance and in which order these enhancements should occur. All the background knowledge required to make these modifications is likely to have a negative impact on the amount of time required for development. To support the learning process associated with framework application instantiation, developers may use different resources, such as framework documentation and code examples. However, these resources may not exist or, when they do exist, they may be incomplete or insufficient for a developer's needs [2].

In this context, techniques for mining software repositories have been applied to acquire knowledge that may be useful for developers when instantiating a framework [3] [4]. The use of mining techniques in software repositories was motivated by the rich amount of information that these repositories could provide about the development process, such as artifacts created or modified, activities performed, and people involved

in the process. This paper proposes the use of process mining techniques to extract information from software repositories about the instantiation process used to create applications. The goal is to mine the instantiation process in terms of what reuse actions were executed to develop a set of applications instantiated from a single framework. The term reuse action refers to object-oriented actions such as sub-classing, implementing an interface or redefining a method. Moreover, in order to have a process, it is also necessary to find existing dependencies such as the order in which the reuse actions are executed or if two or more actions can co-exist. It is expected that this process acts both as part of the framework documentation and as a guide for the re-user during the instantiation process. It is also important to mention that the final framework instantiation process specification will be delivered as an RDL (Reuse Description Language) [5] script, which is a language designed to represent object-oriented framework instantiation processes.

This paper is structured as follows. After an introduction to the research, Section 2 presents the underlying concepts of the solution. Section 0 provides details of the proposed solution. The application of the proposed solution using Eclipse GEF framework is given in Section 4, followed by a comparison with related work in Section 5, and then some conclusions in Section 6.

2 Underlying Concepts

2.1 Reuse Process Specification

RDL (Reuse Description Language) was developed to provide a systematic way of representing the instantiation process for a framework. As shown in Code 1, RDL supports the description of the steps needed for instantiating a framework such as the extension of classes (`CLASS_EXTENSION`) and methods (`METHOD_EXTENSION`). Moreover, an RDL script can specify the sequence in which the activities must be performed, using conditionals (`IF-THEN-ELSE`) and repetition (`LOOP`) commands.

```
1. COOKBOOK ShapesProduct
2. RECIPE main {
3.     packA  = NEW_PACKAGE(FrameworkModel,
4.    "org.reusetool.example.myshapeseditor") ;
5.
6.     LOOP ("Create another shape? "){
7.         // Define Shape SubClass
8.         shapeClass = CLASS_EXTENSION(Shape, packA,"?");
9.         // Refine Abstract Methods
10.        m = METHOD_EXTENSION(Shape,shapeClass,addConnectionAnchor);
11.        m = METHOD_EXTENSION(Shape,shapeClass,getFigure);
12.        m = METHOD_EXTENSION(Shape,shapeClass,getIcon);
13.    }
14. }
```

Code 1. RDL code snippet.

RDL is part of the environment called ReuseTool that supports the reuse of object-oriented frameworks. This support is needed as some of the reuse actions require re-user intervention, such as the one shown in Code 1 line 8, where the name of the subclass *Shape* must be provided at runtime [5]. The ReuseTool is capable of generating an UML Model for the Application by manipulating the Framework UML Model according to the commands present in the RDL Script. For this reason, in addition to the RDL script, the ReuseTool needs the Framework UML Model as input.

Currently, the creation of RDL scripts is a manual process performed by the framework developer. However, the use of mining techniques may be used to assist the creation of RDL scripts. To this end, the actions used to create applications along with the information about in which order they should occur need to be retrieved from software repositories. In this work, the types of actions to be extracted are defined by the actions that can be expressed using RDL, such as class extensions, method extensions and interfaces implementation. Commit information can provide ordering information: reuse actions that happened together will be grouped in a single commit, while reuse actions that happened later on will be in subsequent commits.

2.2 Process Mining

Software repositories represent an important source of information about the software development process. Information systems used to support the execution of business processes also generate a vast amount of event data that can provide insights about the processes being executed. Table 1, for example, presents an event log, containing the event data related to two executions of a business process (Register request) for handling requests for compensation.

Case ID	Event ID	Timestamp	Activity	Resource	Cost
1	35654423	30-12-2010:11.02	Register request	Pete	50
1	35654424	31-12-2010:10.06	Examine thoroughly	Sue	400
1	35654425	05-01-2011:15.12	Check ticket	Mike	100
1	35654426	06-01-2011:11.18	Decide	Sara	200
1	35654427	07-01-2011:14.24	Reject request	Pete	200
2	35654641	06-01-2011:15.02	Register request	Pete	50
2	35654643	07-01-2011:12.06	Check ticket	Mike	100
2	35654644	08-01-2011:14.43	Examine thoroughly	Sean	400
2	35654645	09-01-2011:12.02	Decide	Sara	200
2	35654647	12-01-2011:15.44	Reject request	Ellen	200

Table 1. Event log example from [6]

Using information such as that in Table 1, the field of process mining can extract valuable process-related information from event data. There are three main types of process mining: process discovery, conformance checking and enhancement. Process

discovery deals with the control-flow of a business process, i.e., process discovery techniques try to discover a process model that can represent the behavior of the data in event logs. Conformance checking verifies if the designed process corresponds to what happens in reality. The enhancement perspective tries to identify ways of improving the existing process [6]. In this paper, we are interested in the process discovery perspective which will be used to assist in understanding the instantiation process of a framework, thus indicating which reuse actions should be performed and in which order these actions occur.

The event log in Table 1 stores the information about the executions of a process. Each individual process execution corresponds to a process instance and it can be identified in the log by its case id. A process instance contains a set of events where each event refers to a single process activity. To mine the underlying process model, process instances are represented in the form of traces, an ordered set of events. The ordering information between events is necessary in order to discover causal dependencies between them. Table 2 illustrates the trace representation for each process instance in the event log in Table 1.

Case Id	Trace
1	<Register Request, Examine thoroughly, Check Ticket, Decide, Reject Request>
2	<Register Request, Check Ticket, Examine thoroughly, Decide, Reject Request>

Table 2. Trace representation of event log in **Table 1**.

Figure 1 illustrates a process model mined from the traces presented in Table 2. Comparing the event log information and the mined process, it is possible to notice that each event maps to a single activity. Also, the process flow is obtained through the ordering between events in the trace. For example, the activity *register request* is the first activity to be executed in both traces so it is the first activity in the mined process. The same occurs to the activity *reject request* as the end activity in the process. From the traces in Table 2, we can observe that activities "examine thoroughly" and "check ticket" do not need to be in a sequence since, in case 1, "examine thoroughly" appears before "check ticket" and, in case 2, the opposite occurs. That is the reason these activities appear in parallel in the process modeled in Figure 1.

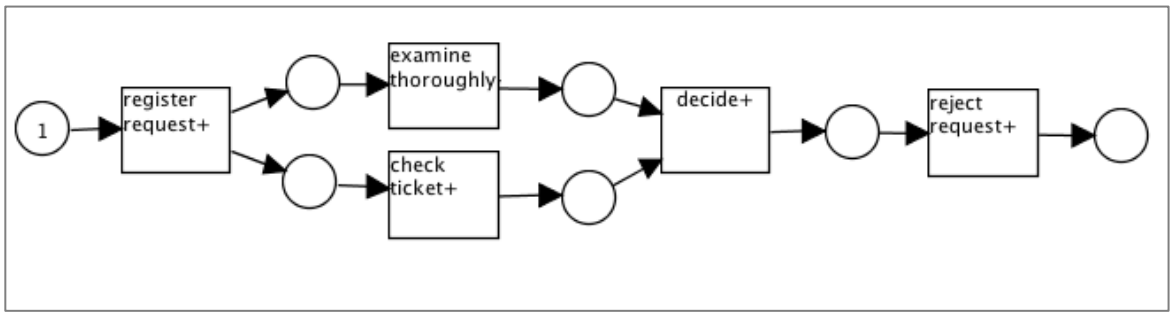

Figure 1. Process model discovered from the event log in **Table 1**

There is a wide variety of process discovery algorithms in the literature, such as the α-algorithm and its successors (α+, α++, β), the fuzzy miner, the heuristic miner and the genetic miner [7]. These algorithms search for workflow patterns in the event log, such as the ones illustrated in Figure 2. Figure 2 patterns can be observed in the process shown in Figure 1. Many of these algorithms were implemented as a plugin for the ProM framework, an environment developed to foment the development of process mining techniques.

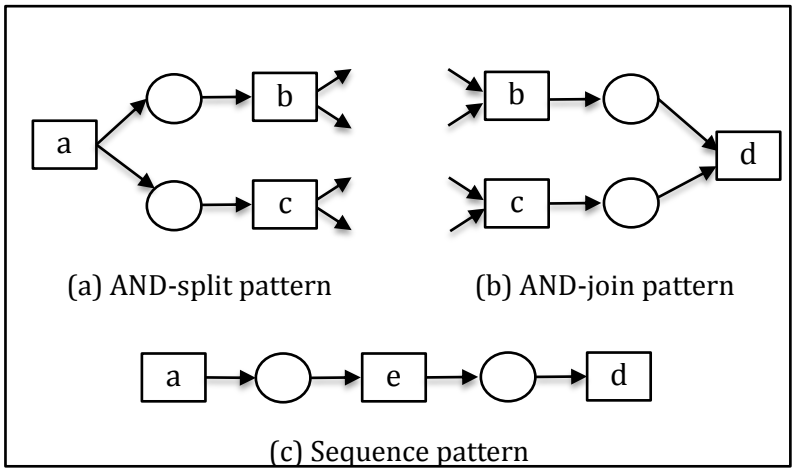

Figure 2. Workflow Patterns examples [6].

3 Proposed Approach

In this paper, we are investigating if for a given framework F and a set of applications FA built from F it is possible to find an instantiation process capable of describing the steps needed to create new applications from F. Besides having a process that represents how applications should be built, we also want this process to be executable. To accomplish these goals, process mining techniques will be applied to the information in software repositories, and the mined instantiation process will be translated to RDL. Figure 3 gives an overview of the steps in the solution.

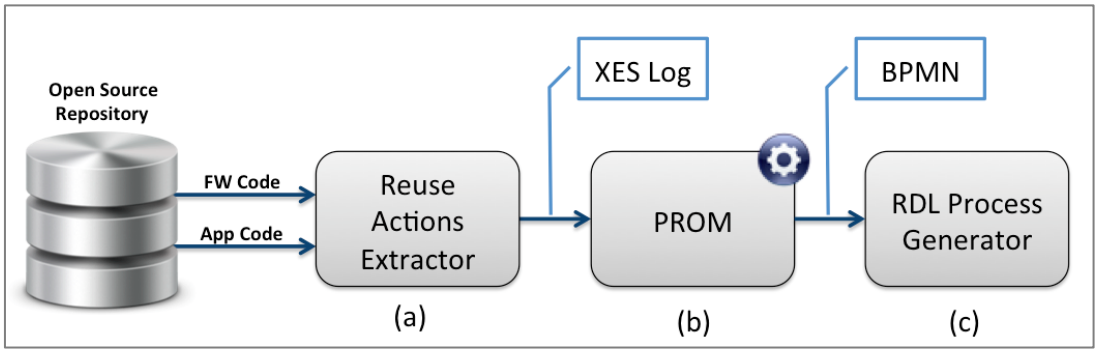

Figure 3. Solution Overview

Process discovery algorithms use an event log as input. To mine the instantiation process the first step consists of constructing the framework event log containing the application's implementation history. This step is represented as Box (a) in Figure 3. The framework and application source code are both analyzed to identify which reuse actions were performed to instantiate the framework. Commit information is also used to help reconstruct the application creation history since it provides information such as which reuse actions were performed at the same time and which actions occurred sequentially. To use the event log structure explained in section 2.2, the framework event log is structured as follows:

1. Each event log stores information about the instantiation process of a single framework;
2. Each framework application corresponds to a process instance;
3. The set of events in a process instance contains the reuse actions performed to build the corresponding application.

The current standard for representing event logs in process mining is called XES (Extensible Event Stream). XES is an XML-based standard which purpose is to provide a general format for interchange event log data between tools and application domains [8]. Code 2 illustrates how the event log data is structured using the XES format. A XES file stores information about a single event log, represented by the *log* tag. The information about the process instances is underneath the *trace* tag. In the example in Code 2, the log contains a single *trace* that has the following attributes: a *name* (Code 2 line 6), an *id* (Code 2 line 7) and a set of events represented by the *event* tags. The attributes of each event are presented underneath the *event* tag.

Once the event log is ready, the process discovery task is executed using process discovery algorithms implemented in the ProM framework [9], as shown in Box (b) of Figure 3. The algorithms in the ProM framework are represented by individual plugins, and two of them are currently being used for the process discovery task. The first one mines a Causal Net [10] using the heuristic miner while the second one converts the Causal Net into a BPMN (Business Process Modeling Notation) process. The BPMN process can be exported from ProM in an XPDL file.

The XPDL (XML Process Definition Language) file describes the mined process in terms of activities and transitions. There are special types of activities called Gateways whose role in the process is to determine the process flow. The last step of the proposed approach – Box (c) of Figure 3 – is concerned with the description of the information in the XPDL file using RDL. To do so, it is necessary to identify conditionals (`IF-THEN-ELSE`) and repetition of activities (`LOOP`). This identification is performed using the DFS (Depth-first Search) algorithm, an algorithm to traverse or search a tree or graph structure. This step is still in progress.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <log xes.version="1.0" xes.features="nested-attributes"
3      openxes.version="1.0RC7" xmlns="http://www.xes-standard.org/">
4      <string key="concept:name" value="Framework process" />
5      <trace>
6          <string key="concept:name" value="vainolo" />
7          <id key="identity:id" value="A43402C8-E867-4744-BF34-E03466D3D486" />
8          <event>
9              <id key="identity:id" value="FBB54B29-1972-45BC-BE97-5B332A3A9926" />
10             <string key="lifecycle:transition" value="complete" />
11             <string key="concept:name" value="Register Request" />
12             <date key="time:timestamp" value="2013-11-30T21:32:25.999-02:00" />
13         </event>
14         <event>
15             <id key="identity:id" value="28C0119D-4F70-46F8-A947-3D54D076B2D7" />
16             <string key="lifecycle:transition" value="complete" />
17             <string key="concept:name" value="Check Ticket" />
18             <date key="time:timestamp" value="2013-11-30T21:32:25.999-02:00" />
19         </event>
20         .
21         .
22         .
23     </trace>
24 </log>
```

Code 2. Example log represented using the XES standard

4 Exploratory Case Study – The GEF Framework

In order to better expose the proposed approach to automatic framework documentation we have conducted an exploratory case study with a popular object-oriented framework implemented in Java, the GEF Framework. The Eclipse Graphical Editing Framework (GEF) [11] is a model-view-controller framework dedicated to creating graphical editors and views for the eclipse UI workbench. Five GEF applications were used to produce the input event log, four example applications found on GEF Git repository [13] (shapes, flow, digraph1 and digraph2) and an application found on GitHub that teaches developers on how to use the GEF framework to create graphical editors [14].

The log creation consists in the analysis of the source code searching for reuse actions. It is worth noting that the set of reuse actions is limited to the reuse actions that can be expressed using RDL, such as class extension, method extension and interfaces implementation. In this exploratory study, the set of reuse actions is formed only by class extension tasks. Figure 4 shows an excerpt from the resulting log. In this log, the *trace* tag represents an application (a process instance). The trace shown in the excerpt corresponds to the application called vainolo (line 6). The *event* tags correspond to reuse actions performed in building the application and the tags underneath them represent the event attributes. The first event in the trace, for example, corresponds to a Class Extension command of the Framework class called *GraphicalEditorWithFlyoutPalette* (lines 11 and 12).

In order to discover the control-flow perspective of a process, process mining techniques need the events in a trace to be ordered. To obtain the events ordering, commit information is taken into account. First, all commits made to build the application were

retrieved. Then, a static analysis of the source code of each commit is made in chronological order. All reuse actions are included in the application trace, except if the reuse action was performed in an earlier commit, i.e., for a given reuse action RA in commit C_i, RA is only included in the application trace if $RA \not\subset C_{i-1}$.

The event log is imported into ProM framework where it can be used as input for process discovery algorithms. Before running the process discovery algorithm, the log is preprocessed to remove noisy behavior. In the context of process mining, noisy behavior corresponds to those events that are infrequently and do not represent the mainstream behavior [6]. The resulting event log contains 62 events, as summarized in Table 3. The first column in Table 3 lists all unique events found in the event log and the other columns indicate the number of times that the corresponding event took place in the application development process.

Currently, the Heuristic Miner plugin is being used for the process discovery task. This plugin produces a process model from the data in the log and represents the model using a causal net as shown in Figure 5. A Causal net is a process representation in the form of a graph where nodes represent process activities and edges correspond to causal dependencies [6]. So, each event in the event log corresponds to a node in the mined process and the edges between then are causal dependencies. Conditionals and loops can be observed in the flow of the mined process. Boxes (a), (b) and (c) in Figure 5, for example, highlight loops that can be found in the process.

```
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="1.0" xes.features="nested-attributes"
    openxes.version="1.0RC7" xmlns="http://www.xes-standard.org/">
    <string key="concept:name" value="Framework process" />
    <trace>
        <string key="concept:name" value="vainolo" />
        <id key="identity:id" value="A43402C8-E867-4744-BF34-E03466D3D486" />
        <event>
            <id key="identity:id" value="FBB54B29-1972-45BC-BE97-5B332A3A9926" />
            <string key="lifecycle:transition" value="complete" />
            <string key="concept:name"
                value="CLASS_EXTENSION_org.eclipse.gef.ui.parts.GraphicalEditorWithFlyoutPalette" />
            <date key="time:timestamp" value="2013-11-30T21:32:25.999-02:00" />
        </event>
        <event>
            <id key="identity:id" value="28C0119D-4F70-46F8-A947-3D54D076B2D7" />
            <string key="lifecycle:transition" value="complete" />
            <string key="concept:name"
                value="CLASS_EXTENSION_org.eclipse.gef.palette.PaletteRoot" />
            <date key="time:timestamp" value="2013-11-30T21:32:25.999-02:00" />
        </event>
        <event>
            <id key="identity:id" value="2B954951-F86F-4984-AE27-2BD88BD1C171" />
            <string key="lifecycle:transition" value="complete" />
            <string key="concept:name" value="CLASS_EXTENSION_org.eclipse.gef.commands.Command" />
            <date key="time:timestamp" value="2013-11-30T21:32:25.999-02:00" />
        </event>
        .
        .
        .
```

Figure 4. An excerpt from the event log

Reuse Action Event	Vainolo	Shapes	Flow	Digraph1	Digraph2
GraphicalEditor				1	1
GraphicalEditorWithFloutPallete	1	1			
Command	5	6	8		
AbstractGraphicalEditPart	3	2	1	2	2
AbstractConnectionEditPart	1	1	1		1
AbstractTreeEditPart		2			
GraphicalNodeEditPolicy	1				
XYLayoutEditPolicy	1	1		1	1
ComponentEdiPolicy	1	1	1		
ConnectionEditPolicy	1		1		1
GraphicalNodeEditPolicy			1		
ContainerEditPolicy			2		
FlowLayoutEditPolicy			2		
AbstractTool				1	1
ActionBarContributor	1	1	1		
ContextMenuProvider		1	1		
Total of Events	**15**	**16**	**19**	**5**	**7**

Table 3. Event log summary

Once the causal net is mined, it is then converted to BPMN and the BPMN process is exported from ProM as an XPDL file. The XPDL file can be used to produce the RDL script in Code 3. As mentioned before, RDL has a `LOOP` command that allows the repetition of a reuse task. Code 3 shows three loops (lines 12-14, 18-20, 29-31) corresponding to the RDL representation of boxes (a), (b) and (c) in Figure 5, respectively. The routing paths in the BPMN process generate the `IF-THEN-ELSE` statements. For example, except in case of Flow application, all applications extend either the class `GraphicalEditor` or the class `GraphicalEditorWithFloutPallete` and this is expressed by the `IF-THEN-ELSE` statement in lines 9-15.

In order to generate the application UML model, besides the mined RDL script, the framework UML model is required. A tool called TopCased was used to obtain GEF UML Model from the framework java code [17]. With both inputs available, the ReuseTool is able to guide the re-user in the creation of new applications for the GEF framework.

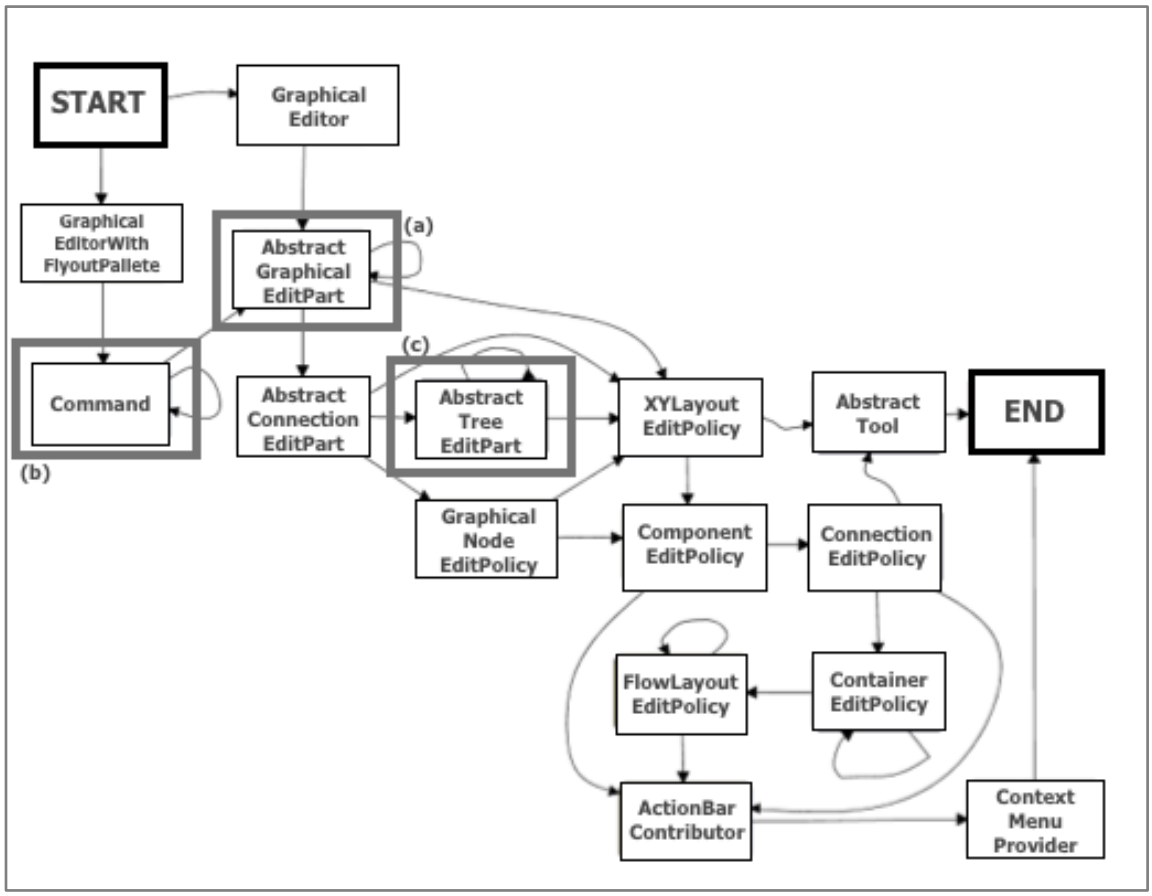

Figure 5. Causal net mined in ProM

The final RDL script was executed to verify if it was capable of generating the UML model for the applications in the event log. As a result, some inconsistencies were found like the extension of classes that the original application did not implement and the absence of classes that should be in the application. This can be a result of the large number of BPMN gateways present in the BPMN model exported from ProM. Currently, graph algorithms are being used to explore the BPMN process and generate the RDL script. Improvements to the RDL script generation are being investigated. Code 4 shows the UML model generated after executing the RDL script following the steps to produce the digraph1 example application. This application could be created with no differences from the original one, contemplating `CLASS_EXTENTION` tasks only.

```
import 'models/gef.uml';
export 'models/app.uml';

COOKBOOK GEFProducts
RECIPE main() {
    appPack = NEW_PACKAGE(appmodel,"?");

    IF ("Create GraphicalEditor?") {
        CLASS_EXTENSION(org.eclipse.gef.ui.parts.GraphicalEditor, appPack, "?");
    } ELSE {
        CLASS_EXTENSION(org.eclipse.gef.ui.parts.GraphicalEditorWithFlyoutPalette, appPack, "?");
        LOOP ("Create new Command?") {
            CLASS_EXTENSION(org.eclipse.gef.commands.Command, appPack, "?");
        }
    }

    CLASS_EXTENSION(org.eclipse.gef.editparts.AbstractGraphicalEditPart, appPack, "?");
    LOOP ("Create new AbstractGraphicalEditPart?") {
        CLASS_EXTENSION(org.eclipse.gef.editparts.AbstractGraphicalEditPart, appPack, "?");
    }

    IF ("Create AbstractConnectionEditPart?") {
        CLASS_EXTENSION(org.eclipse.gef.editparts.AbstractConnectionEditPart, appPack, "?");

        IF ("Create GraphicalNodeEditPolicy?") {
            CLASS_EXTENSION(org.eclipse.gef.editpolicies.GraphicalNodeEditPolicy, appPack, "?");
        }

        LOOP ("Create new AbstractTreeEditPart?") {
            CLASS_EXTENSION(org.eclipse.gef.editparts.AbstractTreeEditPart, appPack, "?");
        }
    }

    IF ("Create XYLayoutEditPolicy?") {
        CLASS_EXTENSION(org.eclipse.gef.editpolicies.XYLayoutEditPolicy, appPack, "?");
    }

    .
    .
    .
    }
END_COOKBOOK
```

Code 3. GEF mined RDL Script

5 Related Work

In software engineering, data mining algorithms can be used to support a wide variety of tasks, including software reuse, which has been supported in different ways. Eclipse Code Recommenders plugin [15] provide recommendations on the APIs developers should use, and it also provides extended documentation for the framework. The research reported in [16] has a different approach; it focuses on the framework concepts. Templates are extracted that provide the steps for implementing a framework concept, such as which packages to import, which classes to extend and which methods to call during the implementation. Spotweb [4] supports software reuse by identifying flexibility points of the framework (hotspots) and the areas of the framework that are rarely used (coldspots). For the re-user, Spotweb offers hotspots hierarchies, and it can recommend code snippets that help the user in the implementation task.

The main difference between the previously mentioned related work and this research is the adoption of a process-oriented approach. Although the work presented in

[16] does have some process notions by presenting a tutorial-like template, it does not provide support for the developer throughout the instantiation process since the template is specific for a given feature of the framework.

In terms of process, the idea of process discovery was first applied in software engineering in [18]. In that work, the objective was to use event-data to discover a software process, i.e., to obtain a formal process capable of describing the activities that should be performed in order to design, develop and maintain a software system. The work presented in this paper has a narrower scope, focusing in software reuse.

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="20110701" xmlns:xmi="http://www.omg.org/spec/XMI/20110701"
   xmlns:uml="http://www.eclipse.org/uml2/4.0.0/UML" xmi:id="_73dcEPgaEeC0HvqCO384og" name="appmodel">
  <packagedElement xmi:type="uml:Package" xmi:id="_Fg_1wWWEEeOMV8PRFmmg2w" name="digraph1">
    <packagedElement xmi:type="uml:Class" xmi:id="_JU4-IGWEEeOMV8PRFmmg2w" name="AbstractGraphicalEditPart1">
      <generalization xmi:id="_JU4-IWWEEeOMV8PRFmmg2w">
        <general xmi:type="uml:Class" href="models/gef.uml#_Qh-nIFezEeOQ15YI6eURGQ"/>
      </generalization>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_J7BxEGWEEeOMV8PRFmmg2w" name="AbstractGraphicalEditPart2">
      <generalization xmi:id="_J7BxEWWEEeOMV8PRFmmg2w">
        <general xmi:type="uml:Class" href="models/gef.uml#_Qh-nIFezEeOQ15YI6eURGQ"/>
      </generalization>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_P5esgGWEEeOMV8PRFmmg2w" name="MyXYLayoutEditPolicy">
      <generalization xmi:id="_P5fTkGWEEeOMV8PRFmmg2w">
        <general xmi:type="uml:Class" href="models/gef.uml#_QiuOAFezEeOQ15YI6eURGQ"/>
      </generalization>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="_SWF44GWEEeOMV8PRFmmg2w" name="MyAbstractTool">
      <generalization xmi:id="_SWF44WWEEeOMV8PRFmmg2w">
        <general xmi:type="uml:Class" href="models/gef.uml#_QmSfYFezEeOQ15YI6eURGQ"/>
      </generalization>
    </packagedElement>
  </packagedElement>
</uml:Model>
```

Code 4. UML Model for digraph1 example application

6 Conclusion

This paper proposes a solution for supporting framework reuse based on the application of process discovery techniques using information available in software repositories. The support is provided in the form of an RDL process that represents the instantiation process of a framework. In this context, this work has two main contributions: (1) the discovery of a process that can be used both as a template for building new framework applications and as part of the framework documentation, and (2) the possibility of executing the process because of its specification in RDL.

A preliminary evaluation of the methods described in this paper indicates that they provide substantial insight and documentation into how applications are produced from a framework. Thus, they can be used effectively to guide and simplify further application development. A more thorough evaluation of the approach will be conducted as the next steps in this research.

References

1. R. E. Johnson and B. Foote, “Designing reusable classes,” Journal of Object-Oriented Programming, 1998.
2. Robillard, M. P. (2009) “What Makes APIs Hard to Learn? Answers from Developers”, IEEE Software, vol. 26, no 6, pp. 27–34.
3. Bruch, M., Mezini, M., and Monperrus, M. (2010) “Mining subclassing directives to improve framework reuse”, 7th IEEE Working Conference on Mining Software Repositories (MSR), pp. 141 –150, Cape Town, 2-3 May.
4. Thummalapenta, S. and Xie, T. (2008) “SpotWeb: Detecting framework hotspots and coldspots via mining open source code on the web”. In: *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 327–336.
5. Oliveira, T. C., Alencar, P. and Cowan, D. (2011) “ReuseTool-An extensible tool support for object-oriented framework reuse”, Journal of Systems and Software, vol. 84, n. 12, pp. 2234–2252.
6. Van der Aalst, W. (2011) Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer.
7. De Weerdt, J., De Backer, M. and Vanthienen, J. (2012) “A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs”, Information Systems, vol. 37, n. 7, pp. 654-676.
8. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 60–75. Springer, Heidelberg (2011)
9. Van Dongen, B. F. (2005) “A Meta Model for Process Mining Data”, In: Proceedings of the CAiSE'05 Workshops (EMOI-INTEROP Workshop), vol. 2, FEUP, Porto, Portugal, 2005, pp. 309-320.
10. Van der Aalst, W. M. P., Adriansyah, A. and Van Dongen, B. (2011) Causal nets: a modeling language tailored towards process Discovery. In: *Proceedings of the 22nd international conference on Concurrency theory*, pp.28-42, Berlin, Heidelberg.
11. Moorel B., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P. (2004) Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework.
12. GitHub. Available in: https://github.com/. Last Access: November 30, 2013.
13. GEF Git repository. Available in: http://git.eclipse.org/c/gef/org.eclipse.gef.git/tree/ Last Access: November 30, 2013.
14. Vainolo – JJTV5_gef – Creating graphical editors with Eclipse and GEF. Available in: https://github.com/vainolo/JJTV5_gef. Last Access: November 30, 2013.
15. Eclipse Code Recommenders. Available in: http://www.eclipse.org/recommenders/. Last access: November 30, 2013.
16. Heydarnoori, A., Czarnecki, K. and Bartolomei, T. T. (2009) “Supporting Framework Use via Automatically Extracted Concept-Implementation Templates”. In: Proceedings of the 23rd European Conference on ECOOP 2009, pp. 344–368, Berlin, Heidelberg.
17. Topcased. Available in: http://www.topcased.org/. Last Access: December 14, 2013.
18. Cook, J. E. and Wolf, A. L. (1998) “Discovering Models of Software Processes from Event-based Data,” ACM Trans. Softw. Eng. Methodol., vol. 7, no. 3, pp. 215–249, Jul. 1998.