Repairing Business Process Models as Retrieved from Source Code

María Fernández-Ropero[1,2], Hajo A. Reijers[1,3], Ricardo Pérez-Castillo[2] and Mario Piattini[2]

[1]Department of Mathematics and Computer Science, Eindhoven University of Technology
Den Dolech 2, 5612 AZ Eindhoven, The Netherlands
{m.fernandezropero, h.a.reijers}@tue.nl
[2]Instituto de Tecnologías y Sistemas de la Información, University of Castilla-La Mancha
Paseo de la Universidad 4, 13071 Ciudad Real, Spain
[3]Perceptive Software
Piet Joubertstraat 4, 7315 AV Apeldoorn, The Netherlands
{marias.fernandez, ricardo.pdelcastillo, mario.piattini}@uclm.es

Abstract. The static analysis of source code has become a feasible solution to obtain underlying business process models from existing information systems. Due to the fact that not all information can be automatically derived from source code (e.g., consider manual activities), such business process models may not entirely reflect the original behavior of the organization. This paper provides a technique to repair such business process models on the basis of event logs collected during the execution of information systems. The technique detects missing sequence flows regarding the event log and tidily adds these sequence flows to the target business process model. In order to enhance its applicability, this technique is tool-supported. Additionally, this paper provides a case study with a real-life system to demonstrate its feasibility.

Keywords: process models, source code mining, event logs, repairing

1 Introduction

Business process management enables organizations to become more efficient, more effective and more readily adaptable to changes than traditional, functional management approaches. Business processes describe the organization's operations, as well as the roles and resources involved [1]. Sometimes, however, business processes models do not explicitly exist in an organization. And even if an organization has created models of its business processes, these could be outdated and misaligned with the actual activities. In cases where business activities are supported by information systems, reverse engineering techniques can be used to obtain business process models from these. This is often an attractive practice, since existing information systems may embed business logic in their source code. For this reason, *business process archeology* has emerged as a set of techniques and tools to mine business processes from source code. Source code contains a lot of business knowledge that has been embedded during the information system maintenance. Thus, business process archeology represents a good start point for business experts, requiring less effort than modeling from scratch. One of these techniques is MARBLE [2], which is based on a model-driven approach, and uses the KDM (Knowledge Discovery Metamodel) standard to represent intermediate models.

While the analysis of source code allows the acquisition of embedded knowledge that is not present anywhere else, their application may entail a semantic loss due to the increase of abstraction level [3]. Business process models obtained in this way can therefore be incomplete, could contain irrelevant information, or may even contain ambiguities that decrease their understandability. The improvement of such a processes model is necessary to address these problems, which helps to have them better reflect reality [4]. To enrich the semantics of business process models it is necessary to consider alternative sources from which to extract knowledge. *Event logs* form one such candidate. In an opposite way to business process archeology, process mining techniques aim at obtaining useful information from event logs by means of process discovery, conformance checking and model enhancement [5, 6]. These event logs are recorded by information system such as enterprise resource planning (ERP) or customer relationship management (CRM) systems, among others, i.e., process-aware information systems (PAISs) [7]. Organizations may also operate traditional (non-process-aware) information systems supporting their business processes, which do not record any event during execution.

This paper presents a technique to repair business processes models as obtained by a static analysis of source code by capturing additional information from event logs. To develop the technique two assumptions based on our previous work were taken into account: (1) business process models, capturing a static viewpoint, are obtained by means of MARBLE, an adaptive framework to recover the underlying business process models from legacy information system; and (2) event logs, representing a dynamic viewpoint, are obtained by means of the technique proposed by *Pérez-Castillo et al.* [8] in which event logs are generated from non process-aware systems, which enables a process mining approach. Business process models obtained with these two different techniques display similarities as well as differences. Hence, our proposed approach finds similar tasks in both models in order to detect missing sequence flows by comparing both artifacts, i.e. those sequences flows that can be inferred from the event log but which are not in the initial business process model. The detected, missing sequence flows are incorporated into the target business process model, making it more complete and accurate regarding to the event log. The actual improvement obtained after this repair step is evaluated in the paper through a case study using a real-life information system. The case study's results show that the repaired business models are indeed more accurate and more complete than the initial model as retrieved by reverse engineering.

The remainder of this paper is organized as follows: Section 2 presents related work. Section 3 introduces the proposed approach to repair business processes models using event logs. Section 4 shows some preliminary results provided by the proposed approach using real-life systems. Finally, Section 5 presents the conclusions and directions for future work.

2 Related Work

In the literature, various techniques are described to obtain business process models. Some of these techniques consider dynamic analysis, which obtain process models from the event logs that are recorded during system execution. These logs represent the actual system performance and several algorithms can be used, such as the alpha algorithm proposed by *Van der Aalst et al.*, a genetic algorithm proposed by *De Medeiros et al.*, a heuristics algorithm proposed by *Weijters et al.*, among others, to mine the business process [9-11]. The event logs used by these algorithms are obtained from process-aware information systems (PAISs) [7], i.e., information systems whose nature facilitates the direct registration of events throughout process execution. Although information systems that are not process-aware do not automatically record event logs, such logs can be obtained by hand or by injecting code to trace by techniques as proposed by *Perez-Castillo et al.* [8]. These event logs are generated when the information system is running, and describe which tasks are executed and in what order for a certain time period. The downside of such event logs is that not all functionalities can be captured, i.e. only tasks that have been carried out at the time of executing the injector. That is, if the injector stores the event logs for a year it is not possible to recover the tasks that are executed, e. g., two years back, or it may not be able to recover those tasks that hardly ever occur but are important for the system.

Apart from dynamic analyses, a static analysis has been proposed. Static analysis obtain process models through the syntactical analysis of the source code, e.g. by *Zou et al.* [12]. They developed a framework based on a set of heuristic rules to extract business processes following model-driven development. The framework statically analyzes the legacy source code and applies the rules to transform pieces of source code in business process elements. Although this work is based on the MDA (Model-Driven Architecture) approach, standards as KDM are not considered. *Ghose et al.*[13], in turn, consider other software artifacts as a set of text-based queries in documentation for extracting business knowledge, but the approach is not based on the MDA. *Perez-Castillo et al.*[2], use standards in their approach to obtain process models. They propose MARBLE to obtain a first approximation of business process that is especially useful for organizations that have never modeled their processes, while their legacy information systems do embed knowledge during its maintenance (knowledge that is only present in the source code, not in the documentation). Unfortunately, the retrieved process models have a low abstraction level, being very close to the code level. Furthermore, not all embedded information can be obtained using MARBLE. Thus, the recovered process models involve several challenges to address.

Neither static nor dynamic analysis can obtain the actual and complete contours of business processes in an organization. *Adriansyah et al.* [14] discuss in their work that a retrieved model often does not describe the process executions as observed in reality, e.g., activities in the model are skipped in the log, the log contains events not described in the model or the order execution of the tasks are different. This work compares the process model with an event log of the same process. In follow up to this observation, *Fahland et al.* [4] suggest to repair business process with the recorded event logs. They obtain subprocesses in event logs not being present in the process model and then, insert them where it is missing. This particular work assumes that the process model has been discovered by mining process (using event logs) or by hand. However, this is mostly realistic in PAIS settings. The present paper is focused on a technique to repair business process using event logs that are also suitable for non-PAISs. Thus, this work combines the static and dynamic analysis in order to improve process models.

3 Technique for Repairing Business Process Models

The repair technique combines artifacts obtained from the static and dynamic analyses of existing information systems, i.e., a first sketch of business process models and event logs collected at runtime. The main goal of the technique is to detect missing sequence flows by comparing both artifacts and build an improved business process model containing these sequence flows. The technique has been defined under two assumptions, which are related to the two previously mentioned techniques. Despite these two assumptions, this approach can be adapted to other techniques with which to reverse business process models or obtain event logs.

Assumption 1. One of the assumptions of the repair technique is that the process models are obtained using MARBLE (Modernization Approach for Recovering Business process from LEgacy systems) [2], a framework for obtaining business processes from legacy information systems (LIS for short), focusing on the phase of reverse engineering. MARBLE is based on KDM, which is recognized as an ISO/IEC 19506 standard [15] and allows abstract conceptual representations of the different views of the architecture of legacy information systems. Afterwards, this knowledge is gradually transformed and refined down to the underlying business processes. For this purpose, MARBLE is divided into four levels of abstraction and defines three transformations. In order to achieve optimal business process management, MARBLE represents business processes by means of Business Processes Model and Notation (BPMN) [16]. This notation is a well-known graphical notation and is aimed to be easily understandable by system analysts as well as business analysts.

Assumption 2. The second assumption is that event logs are obtained by the injection of fragments in specific parts of the information system to generate an event log file during system execution, using the event traces injector proposed in [8]. This approach generates event logs in MXML (Mining XML) format from non-process-aware information systems. Although the technique is generic, the supporting tool that is used in this work, Event Traces Injector (ETI), has been designed for object-oriented systems. Event logs are considered as a suitable knowledge source to discover what is really going on in an organization. Each event log is related to a "run" of the process, i.e., a process instance, and provides additional information about the resource executing or initiating the activity, the timestamp, or data elements. Process mining [17] aims at knowledge extraction from event logs available in information systems. Among the available process mining techniques, this paper uses the Heuristic Miner algorithm. The Heuristics Miner proposed by *Weijters et al.* [11] uses a heuristic approach to provide the control flow of the information system from an event log. It is usually applied to real-life data with not too many different events, or for carrying out further analysis in PROM [18].

Fig. 1 shows the sequence of steps carried out to obtain an improved process model ('Process model'). The start points of the technique are the process model and the event logs. The steps are described in Sections 3.1 to 3.4. To facilitate their understanding, a running example will be progressively developed in the mentioned sections. They relate to a real-life information system, in which the technique is applied to Villasante-Lab, a company devoted to the chemical analysis of water and waste water (cf. Section 4).

3.1 Step 1. Obtain info tasks & diagrams

This step analyzes, on the one hand, the business process according the BPMN notation and, on the other hand, event logs according to the MXML notation. In the process model, each diagram (`BusinessProcessDiagram`) contains several tasks, data objects, and inter-connections between these. In event logs, the name of each event corresponds to the name of the class to which it belongs and the name of the method invoked (`nameClass.nameMethod`). The `nameMethod` is considered the task name, while the `nameClass` is considered the diagram name in which the task is contained.

This step obtains which task is included in which diagram. Diagrams are classified as fine-grained or coarse-grained diagrams in order to apply different treatments depending on the type of granularity (e.g., in an object-oriented system, MARBLE transforms some classes in BPMN diagrams and other as tasks inside another diagram while ETI considers each class as a diagram). This classification is made according to a proposed limit. This signifies that if a diagram contains fewer elements than this limit specified as the number of tasks, then that diagram is considered as a fine-grained diagram.

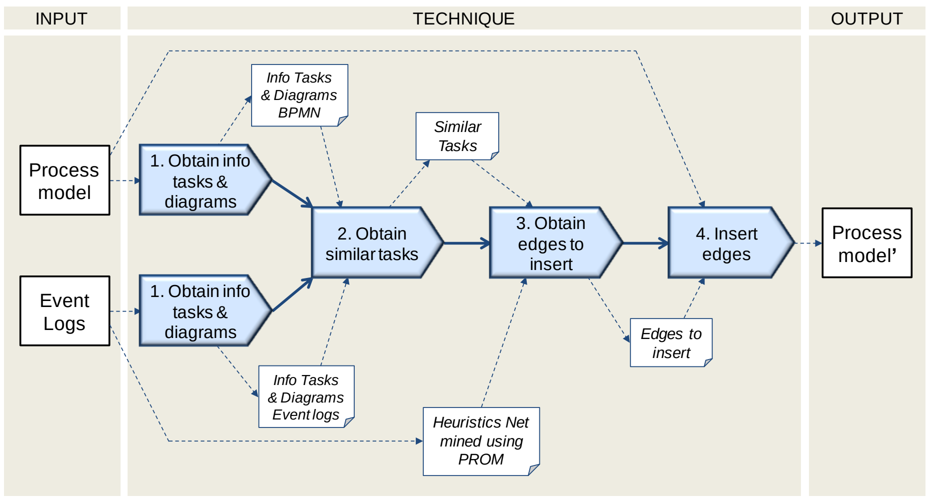
INPUT
TECHNIQUE
OUTPUT
Process model
Event Logs
1. Obtain info tasks & diagrams
1. Obtain info tasks & diagrams
Info Tasks & Diagrams BPMN
Info Tasks & Diagrams Event logs
2. Obtain similar tasks
Similar Tasks
3. Obtain edges to insert
Heuristics Net mined using PROM
Edges to insert
4. Insert edges
Process model'

Fig. 1. Technique to repair BPMN using Event logs

To continue with the running example, Table 1 shows the diagrams obtained after applying both techniques to Villasante-Lab and the number of tasks in each of these. As the tables show, most MXML diagrams are fine-grained and contain very few tasks (usually one) while the BPMN part contains less fine-grained diagrams. Thus, some MXML diagrams correspond (or are equivalent) to tasks in BPMN diagrams. In this running example, the limit to characterize fine-grained diagrams is one task. Besides, a task may contain several occurrences in different diagrams as MXML tasks `getUserManager` and `setInvoiceManager` in Table 1.

3.2 Step 2. Obtain similar tasks.

The information mined from information system using both techniques (MARBLE and Event Traces Injector) displays the following differences:

- **Different types of granularity.** Depending on the extraction techniques, the diagrams show different types of granularity, e.g. in an object-oriented system, MARBLE considers some classes as BPMN diagrams, while other classes are considered tasks inside another diagram, whereas ETI considers classes as diagrams.

Table 1. Extract of tasks Information. BPMN

	Name task	*Name diagram*	*Type Diagram*
BPMN	BaseZoneController	AddPointAdminController	Coarse
	getUserManager	BaseUserController	Coarse
	getClientManager	BaseClientController	Coarse
	initBinder	AnalysisBean	Coarse
	doPrepareView	AnalysisBean	Coarse
	searchZoneNoHistoricas	AnalysisDAO	Coarse
	searchZone	AnalysisDAO	Coarse
	filterUser	AnalysisDAO	Coarse
	convertDissolutionToDissolutionBean	BaseDissolutionController	Coarse
	calculateTotal	BaseDissolutionController	Coarse
	Transform	PdfExport	Fine
	resolveException	ExceptionResolve	Fine
MXML	setZoneManager	BaseZoneController	Fine
	getUserManager	AuthenticationManager	Coarse
	getUserManager	BaseUserController	Coarse
	setRolManager	BaseRolController	Fine
	setInvoiceManager	BaseInvoiceController	Fine
	setInvoiceManager	BaseLinesInvoiceController	Fine
	setDissolutionManager	BaseDissolutionController	Fine
	getClientManager	BaseClientController	Coarse
	initBinder	BaseClientController	Coarse
	searchZoneNoHistoricas	ClientManagerImpl	Coarse
	doHandle	IndexController	Fine
	searchZone	ClientManagerImpl	Coarse

- **Not covering the same number of tasks**. While the BPMN model contains all the business tasks derived from source code, the MXML model only contains those tasks executed during the ETI execution during a certain time. The executed tasks outside the execution period are not recorded and neither are those tasks that rarely occur but are important for the system. Following with the running example, Villasante-Lab, 368 business tasks have been obtained in the BPMN model while only 96 tasks appeared in the MXML model. This represents that the execution of this information system during that time only executed 26% of business tasks of the whole instrumented information system.
- **Similar tasks**. The tasks used in these two models also display similarities. The great challenge is to know which tasks of the MXML model correspond to tasks of the BPMN model (see Fig. 2). This is done by computing the syntactic distance of their name labels. When a MXML task is contained in a fine-grained process, it is necessary to compare the MXML diagram with each BPMN task (due to different granularities) as well as to compare the names of the MXML and BPMN tasks.

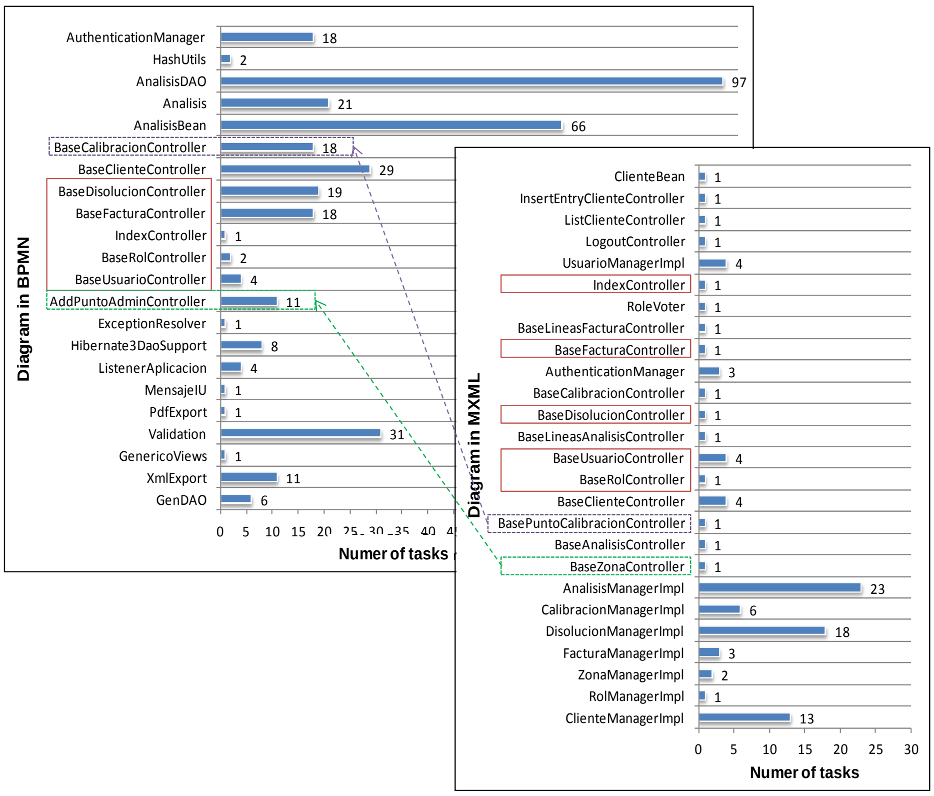

Fig. 2. Comparison BPMN and MXML: Diagrams framed in solid line are similar. Dotted MXML diagrams are tasks in dotted BPMN diagrams

Thus, step 2 selects tasks that are similar in process model and event logs. The step uses the information obtained in the previous step to calculate the distance between two tasks of the two artifacts. If the MXML task is contained in a fine-grained diagram the distance between the BPMN task and MXML diagram is calculated.

The syntactic similarity is calculated using the *Levenshtein* distance [19] of the labels as Algorithms 1 to 3 show.

Algorithm 1: Obtaining Similar Tasks.

```
getSimilarTasks(Info InfoBPMN, Info InfoMXML)
   List similarTasks;
   for(tb:InfoBPPMN.getTasks()) do
      for(tm:InfoMXML.getTasks()) do
         if((getSimilarity(tb,tm))>=LIMIT)then
            similarTasks.add(tb,tm);
         if(tm.getDiagram().getType()==FINE_GRAINE)then
            if ((getSimilarity (tb,tm.getDiagram()))>=LIMIT) then
               similarTasks.add(tb,tm.getDiagram());
   return similarTasks;
```

Algorithm 2: Obtaining the syntactic similarity between task names

```
getSimilarity (Task t1, Task t2)
   double similarity;
   double distance = LevenshteinDistance(t1.name, t2.name);
   similarity = 1 - distance/max(t1.name.length,t2.name.length)
   List adjacentT1 = getAdjacent(t1);
   List adjacentT2 = getAdjacent(t2);
   for(at1: adjacentT1)do
      for(at2: adjacentT2)do
         similarity + =(getSimilarity(at1,at2)/10);
   return similarity;
```

Algorithm 3: Obtaining the similarity between a task and a diagram

```
getSimilarity (Task t, Diagram t)
   double distance= LevenshteinDistance(t.name, d.name);
   return 1 - distance/max(t.name.length,d.name.length);
```

Table 2. Extract of Similar Tasks

BPMN Task	*MXML Task*	*Similarity*
BaseZoneController	**setZoneManager**	1
getUserManager	getUserManager	1
getRolManager	setRolManager	0.923076923
BaseRolController	**setRolManager**	1
BaseInvoiceController	**setInvoiceManager**	1
searchZoneNoHistoricas	searchZoneNoHistoricas	1
searchZone	searchZone	1
filterUser	filterUser	1
searchTypeAnalysis	searchTypeAnalysis	1
searchPointCalibration	searchPointsCalibration	0.956521739
searchLinesDissolution	searchLinesDissolution	1
vote	vote	1
authenticate	authenticate	1

After applying step 2 and following up with the running example, 45 similar tasks were detected, as shown in Table 2. MXML tasks in bold symbolize that these tasks are contained in a fine-grained process which are related to BPMN tasks. In this case, the used limit for the similarity between tasks is 0.9.

3.3 Step 3. Obtain missing sequence flows to be added.

This step uses the Heuristics Net obtained using PROM tool [18] and the set of similar tasks to determine which edges are candidates to be inserted. The source and target of an edge must be in the same diagram in the BPMN. Algorithm 4 shows the procedure used in this step. For each edge, the source and target task are searched from the set of similar tasks (line 5-6). If there are BPMN tasks similar to both tasks (source and target), then the occurrence of BPMN tasks included in the same BPMN diagram are checked (line 11). If the MXML target task has no similar BPMN task (line 14), an intermediate task is then searched (line 15-16). In this case, the obtained edge is induced by transitivity. Similarly, if the MXML source task has no similar BPMN task (line 25), an intermediate task is also searched (line 26-27).

To follow up with the running example, in this step 145 edges are studied, obtaining from Heuristics Net. After applying the third step 14 direct edges and 11 edges are transitively obtained as is shown in Table 3. However, edges with reflexive flows (same source and target) are not inserted in the model since they do not provide additional semantics.

Algorithm 4: Obtaining Edges to insert.

```
1   getSimilarEdges(HeuristicsNet h, similarTasks)
2      List similarDirectEdges;
3      List similarInducedEdges;
4      for(edge: h.getEdges()) do
5         List similarBPMNsources = getBPMNSimilar(edge.source, similarTasks);
6         List similarBPMNtarget = getBPMNSimilar(edge.target, similarTasks);
7         if(!similarBPMNsources.isEmpty())then
8            if(!similarBPMNtarget.isEmpty())then
9               for (Task source: similarBPMNsources) do
10                 for(Task target: similarBPMNtarget) do
11                    if(source.getDiagram()==target.getDiagram() &&
12                     source!=target) then
13                       similarDirectEdges.add(new Edge(source,target));
14           else then
15              for(intermediateEdge: h.getEdges()) do
16                 if(intermediateEdge.source == edge.target) then
17                    List similarBPMNtarget=
18                    getBPMNSimilar(intermediateEdge.target,similarTasks);
19                    if(!similarBPMNtarget.isEmpty())then
20                       for (Task source: similarBPMNsources) do
21                          for(Task target: similarBPMNtarget) do
22                             if(source.getDiagram()==target.getDiagram() &&
23                             source!=target) then
24                                similarInducedEdges.add(new Edge(source,target));
25        else then
```

```
          for(intermediateEdge: h.getEdges()) do
            if(intermediateEdge.target == edge.source)then
               List similarBPMNtarget=getBPMNSimilar(intermediateEdge.target,
               similarTasks);
               if(!similarBPMNtarget.isEmpty())then
                  for (Task source: similarBPMNsources) do
                    for(Task target: similarBPMNtarget) do
                       if(source.getDiagram()==target.getDiagram() &&
                       source!=target) then
                           similarInducedEdges.add(new Edge(source,target));
      return similarDirectEdges, similarInducedEdges;
```

3.4 Step 4. Insert missing sequence flows.

In the last step, the edges obtained in the previous step (see Table 3) are added to the process model. For each edge, its source task and its target task are located in the diagram and the sequence flow between both of these does not exist, the sequence flow is added. In the running example, 25 sequence flows (SF) are inserted in the process model since none of these previously existed.

Table 3. Sequence Flows to insert

	BPMN Source Task	*BPMN Target Task*	**BPMN Diagram**
Direct Sequence Flows	getAnalysisManager	BaseAnalysisController	AnalysisBean
	BaseAnalysisController	getAnalysisManager	AnalysisBean
	getCalibrationManager	BasePointCalibrationController	BaseCalibrationController
	BasePointCalibrationController	getCalibrationManager	BaseCalibrationController
	doHandle	doPrepareView	AnalysisBean
	initBinder	doHandle	AnalysisBean
	getDissolutionManager	BaseDissolutionController	BaseDissolutionController
	BaseDissolutionController	getDissolutionManager	BaseDissolutionController
	getInvoiceManager	BaseInvoiceController	BaseInvoiceController
	BaseInvoiceController	getInvoiceManager	BaseInvoiceController
	getRolManager	BaseRolController	BaseRolController
	BaseRolController	getRolManager	BaseRolController
	getZoneManager	BaseZoneController	AddPointAdminController
	BaseZoneController	getZoneManager	AddPointAdminController
Transitive Sequence Flows	PaginateAnalysisFiltered	searchTypeAnalysis	AnalysisDAO
	insertAnalysis	searchTypeAnalysis	AnalysisDAO
	searchAllClient	searchZone	AnalysisDAO
	searchPointSample	PaginateDissolutionsFiltered	AnalysisDAO
	searchTypeAnalysis	searchAllClient	AnalysisDAO
	searchZone	searchPointSample	AnalysisDAO
	searchCalibration	searchPointCalibration	AnalysisDAO
	searchPointCalibration	searchCalibration	AnalysisDAO
	PaginateDissolutionsFiltered	searchLinesDissolution	AnalysisDAO
	searchLinesDissolution	searchSubstanceReactive	AnalysisDAO
	searchSubstanceReactive	searchSubstanceOfAnalysis	AnalysisDAO

4 Case Study

This section provides a case study concerning Villasante-Lab, in particular the system presented in the running example. The case study has been conducted following the formal protocol developed by *Runeson et al.* [20] for conducting case studies in the software engineering field. The following sections show the stages of this protocol: the design, selection procedure, execution procedure and data collection, analysis and interpretation, and finally, the threats to validity.

4.1 Case study design

The *object* of this study is the proposed repair technique, while the *purpose* is the evaluation of its effectiveness in a real-life context in terms of accurateness and completeness. The following research questions (RQ) are established in order to carry out the case study:

RQ1: Are repaired business models more accurate than preliminary models obtained by reverse engineering from source code?

RQ2: Are repaired business models more complete than preliminary models obtained by reverse engineering from source code?

The case study follows the *embedded* case study design according to the classification proposed by Yin [21], since the case study consists of multiple units of analysis. The independent variables used in this study are business processes models. As dependent variables, conformance checking techniques are used in order to measure the fit degree between event logs and the target business process model after applying the technique. Conformance checking compares the observed and modeled behavior (i.e., event log). Hence, to answer the question RQ1, the dependent variable is the *fitness* value which is often seen as the most important quality dimension for comparing model and log [17, 22]. The fitness values vary between 0 and 1. A model has a perfect fitness (i.e., 1) if each trace in the event log can be replayed by the process model from beginning to end. To address question RQ2, as independent variable the *density* of the business process model is used, i.e., the ratio of the total number of edges in a process model to the theoretically maximum number of edges. The density, after inserting sequence flows, can only increase therefore this evaluation shows what to extent in a realistic case. RQ1 and RQ2 are therefore evaluated by means of quantitative research together with a qualitative evaluation, which focuses on the effectiveness of the proposed repair technique.

4.2 Case selection procedure

In order to select the case under study the following set of selection criteria are formulated: (1) the system should be a real-life information system currently in production; (2) the size of the system should be greater to 20 KLOC (thousands of lines of source code) to make it more likely that the system under study supports more than a single business process; (3) the system should be written in Java language to be able to use the supporting tools (MARBLE and Event Traces Injector).

After analyzing a dozen of information systems of partner companies according to criteria, the selected case was Villasante-Lab, a web application of 26 KLOC devoted to support operations of a chemical laboratory of the water and waste industry.

4.3 Execution procedure and data collection

The procedure to carry out the case study consists of the following steps according to the proposed technique. Particular details of the execution are shown in the running example developed throughout Section 3.

1. Business process models are mined from the source code using MARBLE.
2. Event logs are obtained using the Event Traces Injector.
3. The repair technique is applied using the artifacts generated according to the described steps. In order to facilitate its execution, the technique has been implemented as a plug-in in the PROM tool.
4. The fitness in both business process models – the original from information and repaired using the proposed technique – is measured using the replayer proposed by *Adriansyah et al.* [22]. This technique is developed as a plug-in in the PROM tool. The fitness value is collected to carry out the conformance checking.
5. After the whole execution, the collected information is statistically analyzed to answer the research questions.

Table 4. Case Study's statistics

BP model	*#tasks*	*Initial*		*Final*		*#inserted SF*	*Density gain*
		Density	*#SF*	*Density*	*#SF*		
GenDAO	6	0.1333	2	0.1333	2	0	0
XmlExport	11	0.0355	60	0.0355	60	0	0
GenericViews	1	0	0	0	0	0	0
Validation	31	0.0448	35	0.0448	35	0	0
PdfExport	1	0	0	0	0	0	0
MessageIU	0	0	0	0	0	0	0
ListenerAplication	4	0.2222	2	0.2222	2	0	0
Hibernate3DaoSupport	8	0.1429	4	0.1429	4	0	0
ExceptionResolve	1	2.0000	0	2.0000	0	0	0
AddPointAdminController	11	0.0175	3	0.0292	5	2	0.0117
BaseUserController	4	0.0667	1	0.0667	1	0	0
BaseRolController	2	0.1667	1	0.5000	3	2	0.3333
IndexController	0	0	0	0	0	0	0
BaseInvoiceController	18	0.1087	28	0.1159	30	2	0.0072
BaseDissolutionController	19	0.1082	25	0.1169	27	2	0.0087
BaseClientController	29	0.1261	80	0.1261	80	0	0
BaseCalibrationController	18	0.0627	27	0.0650	29	2	0.0023
AnalysisBean	66	0.0382	255	0.0386	259	4	0.0004
Analysis	21	0	0	0	0	0	0
AnalysisDAO	97	0.0265	124	0.0279	135	11	0.0014
HashUtils	2	0.6667	0	0.6667	0	0	0
AuthenticationManager	18	0.0342	4	0.0342	4	0	0
TOTAL	368	0.0096	651	0.0100	676	25	0.0004

4.4 Analysis and interpretation

After the full execution of the case study, the values of the fitness were collected for the business process model. Although missing sequence flows were only detected in seven business process diagrams, as Table 3 shows, the fitness was calculated for the whole process model. The results demonstrated that the fitness of the repaired BP model (0.6064) is greater than the original fitness (0.3804), i.e., the repaired model fits 59.41% better to the observed behavior. However, the fitness is not yet close to 1 since, as was shown in Section 3.2: only 26% of business tasks of the whole information system are captured in the event logs.

Table 4 summarizes the statistics of the case study. Once the BPMN, the MXML and the Heuristic Net were available, the total time spent on carrying out the repair was 973 milliseconds. In all the cases the density gain (final density - initial density) was positive, even reaching a 33.33% gain.

Hence, the research question RQ1 may be positively answered owing to the fitness has increased, i.e., the repaired business models are more accurate than the preliminary model obtained by reverse engineering from source code. Similarly, the research question RQ2 may be positively answered since the final model is more connected and therefore more complete.

4.5 Threats to validity

This section presents the threats to the validity of this case study and possible actions to address them. The threats are divided in three types of validity: internal, construct and external validity.

Regarding the internal validity, the study considers a process model and event logs obtained from an information system. However, the study may be replicated by using more information systems, to consider a larger sample of process models. Besides, the support tools (MARBLE and ETI) could be a factor that affects the case study results since the technique depends on the settings of retrieved process model and event logs.

With regard to construct validity, the study considers measures to evaluate the research question. Nevertheless, there are other measures in literature that may be used instead. Hence, additional measures should be evaluated in the future, such as shown in [3]. Another threat to construct validity is the similarity algorithm used in step 2 to obtain similar tasks (Algorithm 1). In order to address this threat, other possible similarity algorithms may be considered as e.g., including the semantic similarity.

Concerning the external validity, this study considers the whole population to be business process models retrieved by reverse engineering from legacy information systems as well as event logs obtained from the same information system. However, the obtained results obtained cannot be strictly generalized to all types of information. This threat may be mitigated by replicating the study using systems implemented in different platforms.

5 Conclusions and Future work

Reverse engineering has become a feasible solution to mine business processes models from existing information systems. Unfortunately, these retrieved business processes models entail some challenges that are necessary to address if synch models form the basis for properly managing these business processes.

Incompleteness is one such important challenge to deal with in a retrieved business processes model, since data are distributed across several sources. Missing sequence flows between elements decreases the understandability of the model since it may not reflect the real behavior of an organization. In order to address this challenge, this paper present a technique for repairing business processes models obtained from information systems using event logs. The technique builds on two assumptions: (1) business process models, which represent the static viewpoint of the organization, are mined by the archeology tool MARBLE, which is an adaptive framework to recover business process models underlying legacy information system; and (2) event logs, which represent the dynamic viewpoint of an organization, are obtained by means of the technique proposed in [8], since event logs cannot automatically be generated from non-process-aware systems. Despite these assumptions, the main ideas of this approach can be easily adapted to other reverse engineering techniques and platforms. In fact, to ensure its feasibility this technique has been validated by means of an industrial case study. The results of this case study show that the fitness of the process model increases after applying the technique, i.e., repairing business process model leads to a more faithful representation of the observed behavior.

Future work will aim at incorporating a mechanism to calculate the semantic distance between two tasks. Besides, with both mechanisms (syntactic and semantic similarity) can be performed a grouping of similar tasks in order to decrease the number of fine-grained tasks, i.e., those tasks that do not perform a real business activity. Finally, a mechanism is called for to detect tasks' labels which are poor in descriptive quality, i.e., those task labels that have several occurrences in the model and do not clearly represent their purpose.

6 Acknowledgments

This work was supported by the FPU Spanish Program and the R&D projects GEODAS (TIN2012-39493-C03-01), PEGASO /MAGO (TIN2009-13718-C02-01) and ARMONIAS (PII2I09-0223-7948). This work has been additionally supported by Eindhoven University of Technology.

7 References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer-Verlag Berlin Heidelberg, Leipzig, Germany (2007)
2. Pérez-Castillo, R., García-Rodríguez de Guzmán, I., Piattini, M.: Business Process Archeology using MARBLE. Information and Software Technology (2011)

3. Fernández-Ropero, M., Pérez-Castillo, R., Caballero, I., Piattini, M.: Quality-Driven Business Process Refactoring. International Conference on Business Information Systems (ICBIS 2012), pp. 960-966, Paris, France (2012)
4. Fahland, D., Aalst, W.M.P.v.d.: Repairing Process Models to Reflect Reality. (2012)
5. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
6. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A., Song, M., Verbeek, H.: Business process mining: An industrial application. Information Systems 32, 713-732 (2007)
7. Dumas, M., Van Der Aalst, W., Ter Hofstede, A.: Process-aware information systems. Wiley Online Library (2005)
8. Pérez-Castillo, R., Weber, B., García Rodríguez de Guzmán, I., Piattini, M.: Generating Event Logs from Non-Process-Aware Systems Enabling Business Process Mining. Enterprise Information System Journal 5, 301–335 (2011)
9. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16, 1128-1142 (2004)
10. De Medeiros, A.K.A., Weijters, A., Van Der Aalst, W.: Using genetic algorithms to mine process models: representation, operators and results. Beta, Research School for Operations Management and Logistics (2005)
11. Weijters, A., van der Aalst, W.M.P., de Medeiros, A.K.A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP 166, (2006)
12. Zou, Y., Hung, M.: An Approach for Extracting Workflows from E-Commerce Applications. Proceedings of the Fourteenth International Conference on Program Comprehension, pp. 127-136. IEEE Computer Society (2006)
13. Ghose, A., Koliadis, G., Chueng, A.: Process discovery from model and text artefacts. In: Services, 2007 IEEE Congress on, pp. 167-174. IEEE, (Year)
14. Adriansyah, A., van Dongen, B., Van der Aalst, W.: Conformance checking using cost-based fitness analysis. pp. 55-64. IEEE, (Year)
15. ISO/IEC: ISO/IEC 19506:2012. Information technology -- Object Management Group Architecture-Driven Modernization (ADM) -- Knowledge Discovery Meta-Model (KDM), pp. 331. ISO/IEC (2012)
16. http://www.omg.org/spec/BPMN/2.0/PDF/
17. van der Aalst, W.: Process Mining: Overview and Opportunities. ACM Transactions on Management Information Systems (TMIS) 3, 7 (2012)
18. Promtools.org: ProM Tool. (2010)
19. Lcvenshtcin, V.: BINARY coors CAPABLE or ‘CORRECTING DELETIONS, INSERTIONS, AND REVERSALS. In: Soviet Physics-Doklady. (Year)
20. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Eng. 14, 131-164 (2009)
21. Yin, R.K.: Case study research. Design and methods. Sage, London (2003)
22. van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery (2012)