Toward Obtaining Event Logs from Legacy Code

Ricardo Pérez-Castillo[1], Barbara Weber[2], Ignacio García-Rodríguez de Guzmán[1] and Mario Piattini[1]

[1] Alarcos Research Group, University of Castilla-La Mancha
Paseo de la Universidad, 4 13071, Ciudad Real, Spain
{ricardo.pdelcastillo, ignacio.grodriguez, mario.piattini}and@uclm.es
[2] University of Innsbruck
Technikerstraße 21a, 6020 Innsbruck, Austria
barbara.weber@uibk.ac.at

Abstract. Information systems are ageing over time and become legacy information systems which often embed business knowledge that is not present in any other artifact. This embedded knowledge must be preserved to align the modernized versions of the legacy systems with the current business processes of an organization. Process mining is a powerful tool to discover and preserve business knowledge. Most process mining techniques and tools use event logs, registered during execution of process-aware information systems, as the key source of knowledge. Unfortunately, the majority of traditional information systems is not process-aware and does not have any built-in logging mechanisms. Thus, this paper defines the main challenges to be addressed as well as a preliminary solution to obtain event logs from traditional systems. The solution consists of a technique that statically analyzes the source code and modifies it in a non-invasive way. Finally, the modified source code enables the event log registration at runtime based on dynamic source code analysis.

1 Introduction

Business processes have become a key asset in organizations, since processes allow them to know and control their daily performance, and to improve their competitiveness [2]. Thereby, information systems automate most of the business processes of an organization [14]. However, due to uncontrolled maintenance information systems are ageing over time and become legacy information systems (LIS) [11]. They gradually embed meaningful business knowledge that is not present in any other asset of the organization [7]. When maintainability of LISs diminishes below acceptable limits, they must be replaced by improved versions [8]. To ensure that the new system is aligned with the organization's business processes, the embedded business knowledge needs to be preserved [5]. The business knowledge preservation requires an in-depth understanding of how the information systems currently support the organization's business processes. This problem motivates the use of process mining, which became a powerful tool to understand what is really going on in an organization by observing the information systems [12].

Usually, event logs are obtained from Process-Aware Information Systems (PAIS) [3], i.e., whose nature facilitates the registration of events throughout process execution. Indeed, most process mining techniques and tools are developed for this kind of information systems [2]. In addition to PAIS, there is a vast amount of

traditional systems that also support the business processes of an organization, and could thus benefit from process mining. Nevertheless, non process-aware systems imply five key challenges for obtain meaningful event logs. This paper proposes a technique for addressing these challenges and for obtaining process event logs from traditional (non process-aware) information systems. The technique is based on both static and dynamic analysis of the source code of the systems. Firstly, the static analysis syntactically analyzes the source code and injects pieces of source code in a non-invasive way in specific parts of the system. Secondly, the dynamic analysis of the modified source code makes it possible to write an event log file in MXML format during system execution. The proposed technique is further supported by specific information provided by business experts and system analysts who know the system.

The remainder of this paper is organized as follows. Section 2 introduces the main challenges for obtaining event logs from traditional information systems. Section 3 presents the proposed technique to tackle these challenges. Section 4 discusses related work and finally, Section 5 provides a conclusion and discusses future work.

2 Process-Awareness Challenges

Challenge 1 - Missing Process-Awareness. The first challenge is to know what business activities are executed. While PAISs manage processes (i.e. a sequence of activities with a common business goal using explicit process descriptions) [14], LIS are a set of methods, functions or procedures (*callable units* in general) where processes are only implicitly described. LIS can be seen as a graph where the nodes are the different callable units, and the arcs are the calls between callable units, i.e., the call graph represents the control flow of a LIS. To address this challenge *Zou et al.* [15] proposed the *"a callable unit / a business activity"* approach, which considers each callable unit as a candidate business activity in a process mining context. This approach provides a good starting point, but ignores other important challenges such as, for example, the different granularity of callable units and activities (Challenge 1) and the mixture of business- related and technical callable units (Challenge 3).

Challenge 2 - Granularity. The different granularity of business activities and callable units in LIS constitutes an important challenge. In [12], each callable unit in a LIS is considered as an activity to be registered in an event log. However, LISs typically contain thousands of callable units, many of which are very fine-grained, not directly supporting any business activity. To avoid that the mined processes get bloated with unnecessary details, too fine-grained callable units should not be considered in the event log. In this sense, different solutions can be implemented to discard fine-grained callable units. On the one hand, source code metrics (such as the lines of source code or cyclomatic complexity metric) could be used to determine if a callable unit is a coarse- or fine-grained unit. On the other hand, heuristics (like discarding *getter* and *setter* methods, or discarding units when call hierarchies reach a specific depth) could offer a good alternative with minimal computational costs.

Challenge 3 - Discarding Technical Code. Challenge 3 is caused by the fact that LISs typically contain several callable units, which cannot be considered as business activities. Callable units can be grouped into two domains: (i) the *problem domain* contains the callable units related to the business entities and functionalities of the LIS (i.e., these units implement the business processes of the organization) and (ii) the

solution domain contains the callable units related to the technical nature of the platform used in the LIS and aids the callable units of the previous group. Since callable units belonging to the solution domain do not constitute business activities, they should not be considered in the event log. Therefore, callable units in charge of auxiliary or technical functions that are not related to any use case of the system could be discarded. However, due to the delocalization and interleaving problems [10], the problem and solution domain groups are not always disjoint sets (i.e., the technical and business code are usually mixed), thus requiring that system analysts provide the information about whether a callable unit belongs to the problem or solution domain.

Challenge 4 - Process Scope. Another challenge is to establish the scope of a business process (i.e., to identify where a process instance starts and ends). While the start and end points of a business process are explicitly defined in PAISs, LIS lack explicit information about the supported processes. Unfortunately, the information where a process starts and ends cannot be automatically derived from the source code, but must be provided by business experts (who know the business processes of the organization as well as their start and end activities) and system analysts (who know what callable units in the source code support the start and end activities).

Challenge 5 - Process Instance Scope. The lack of process-awareness in LIS causes another fundamental challenge which is due to the fact that a business process is typically not only executed once, but multiple instances are executed concurrently. If a particular business activity is executed (i.e., callable unit is invoked), this particular event has to be correctly linked to one of the running process instances. Correlating an activity with a data set, which uniquely identifies the process instance it belongs to, poses significant challenges. In particular, it has to be established which objects can be used for uniquely identifying a process instance (i.e., what the correlation data is). If correlation objects have been identified, the location of these objects in each callable unit has to be determined (i.e., the argument or variable in each callable unit that contains the correlation data). This requires the input of business experts and systems analysts who know the LIS and the processes it supports. Unfortunately, however, there are some units where the selected correlation data is not present. For this, traceability mechanisms throughout callable units are needed to have the correlation data available at any place of the legacy source code.

3 A Preliminary Solution

This paper proposes a technique to obtain event logs from non process-aware systems based on a combination of static and dynamic analysis of source code addressing the discussed challenges. Our proposal presents a generic technique, although it is specially designed for object-oriented systems. The *static analysis* is the key stage of the technique, where special sentences for writing events during system execution are injected in the code. Due to the missing process-awareness of LISs this stage poses several challenges (cf. Section 2). While challenges C1 and C2 can be addressed in a fully automated manner (Task 5 and 6 in Fig. 1), challenges C3, C4 and C5 require input from business experts and system analysts (Task 1 - 4 in Fig. 1).

In Task 1, to deal with the process scope challenge (Challenge 4), business experts establish the start and end business activities of the business processes to be discovered. In parallel, system analysts examine in Task 2 the legacy source code and

filter the directories, files or set of callable units that support business activities (i.e., they select the callable units belonging to the problem domain), thereby reducing potential noise in the event log due to technical source code (Challenge C3). Task 3 is the mapping between start/end business activities and the callable units supporting them, which is again supported by system analysts (Challenge C4).

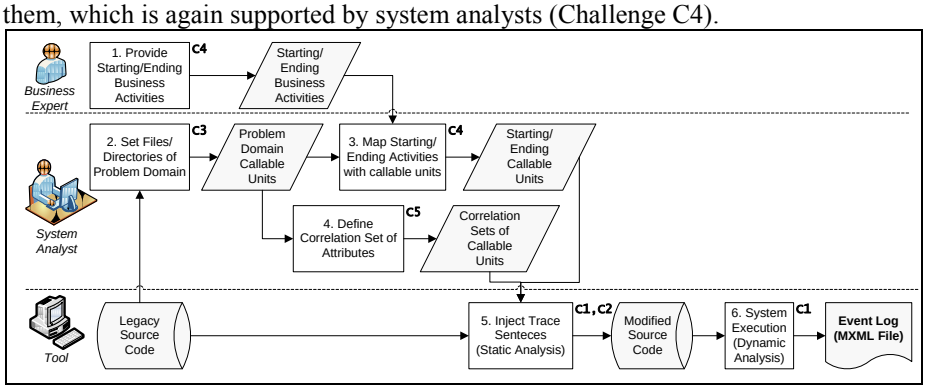

Fig. 1. The overall process carried out by means of the proposed technique

In Task 4 system analysts establish the correlation data set for each callable unit which is uniquely identifying a process instance (Challenge C5). For this, the correlation data is mapped to parameters of each callable unit. This information is then used during run-time to correlate the executed activities with the proper process instance. After that, Task 5 consists of the syntactic analysis of the source code. A parser automatically analyzes and injects on the fly the special sentences writing the event long during system execution. During the static analysis, the source code is broken down into callable units (Challenge 1). All callable units not belonging to the problem domain subgroup selected by the system analyst in Task 3 (Challenge 3) and all fine-grained callable units (e.g., *setter*, *getter*, *constructor*, *toString* and *equals* callable units) are then discarded (Challenge C2). Finally, in each of the filtered callable units, two sentences are injected at the beginning and the end of each respective unit (the first one with a *start* event type, and the second one represents the *complete* event for the same business activity). Moreover, the correlation data defined for the unit as well as information whether or not the unit represents a start or end activity are included in the sentences. When the modified code is executed, the injected sentences invoke a function, which writes the respective event in the log.

The *dynamic analysis* is performed after the static analysis, thus the modified source code can be released to production again. The new code allows to write event log files according to the MXML (Mining XML) format, which is used by the process mining tool *ProM* [13]. When the control flow of the information system reaches an injected sentence, a new event is added to the event log. The events are written by means of a function, which searches the adequate process of the event log where the event must be written using an *Xpath* expression. If the process is null, then a new process is created. After that, the function examines the correlation data to determine to which process instance the event has to be added. If the correlation data is empty, then the function takes the correlation data of the previously executed callable unit to add the event to the correct process instance. This solution is based on simple heuristics and allows correlating events and process instances when no correlation

data is available for the respective event. Moreover, in order to add the event to the correct process instance, the function again uses an *Xpath* expression taking the correlation data into account. If the expression does not find a process instance for the correlation data (i.e., because the event belongs to a start activity), the function creates a new process instance for the correlation data. Finally, when the function has determined the correct process instance, it adds the event to that instance. The event, represented as an *AuditTrailEntry* element in an MXML file, is created with (i) the name of the executed callable unit that represent the *WorkflowModelElement*; (ii) the event type that is also a parameter of the function; (iii) the user of the system that executed the callable unit (or the user of the session if the system is a web application), which represents the *originator* element; and finally (iv) the system date and time when the callable unit was executed to represent the *timestamp* element.

4 Related Work

There are some works related to business processes recovery from non process-aware information systems. *Zou et al* [15] developed a framework to recover workflows from LISs. This framework statically analyzes the source code and applies a set of heuristic rules to discover business knowledge from source code. *Pérez-Castillo et al* [9] make another proposal based on static analysis that uses a set of business patterns to discover business processes from source code. Both approaches solely rely on static analysis, which has the disadvantage that activities cannot be linked correctly to process instances, since the required correlation data is only known at runtime. Thus, other solutions based on dynamic analysis have been suggested. *Cai et al.* [1] propose an approach that combines requirement reacquisition with dynamic analysis. Firstly, a set of use cases is recovered by means of interviewing the system's users. Secondly, the system is dynamically traced based on these use cases to recover business processes. In all these works, the technique for recovering event logs is restricted to a specific mining algorithm. In contrast, our solution proposes a technique based on dynamic analysis (combined with static analysis) to obtain MXML event logs from traditional information systems that is not restricted to a specific process mining algorithm. Similar to our approach the work of *Ingvaldsen et al.* [6] aims at obtaining logs in MXML format from ERP systems. Thereby, they consider the SAP transaction data to obtain event logs. In contrast, our approach aims at traditional information systems without any built-in logging features. In addition, *Günther et al.* [4] provide a generic import framework for obtaining MXML event logs from different PAISs.

5 Conclusions and Future Work

This paper presents a novel technique based on static and dynamic analysis of source code to obtain event logs from non process-aware systems. Thereby, the obtained event log can be used to discover business processes in the same way than an event log obtained from any PAIS. Thus, all the research and development efforts carried out in the process mining field may be exploited for traditional information systems. Achieving this goal is very ambitious since at least five key challenges must be addressed: (i) missing process-awareness, (ii) granularity, (iii) discarding technical code, (iv) process scope and (v) process instance scope.

In a first step, the proposed technique applies static analysis for injecting special sentences in the source code. In a second step, the modified source code is executed, and an event log is written during system execution. In principle, the static analysis of the system has to be performed only once, and then the modified source code can be dynamically analyzed several times to obtain different event logs. However, the feedback obtained by business experts and systems analysts, after the first static and dynamic analysis, can be used to incrementally refine the next static analysis for improving the results obtained during dynamic analysis.

Our work in progress focuses on the improvement of the proposed technique. A traceability mechanism will be implemented taking the call hierarchies into account to deal with lost and scattered correlation data. In addition, to accurately detect the strengths and weakness of the proposal it be validated by means of a case study.

Acknowledgement
This work was supported by the FPU Spanish Program; by the R+D projects ALTAMIRA (PII2I09-0106-2463), INGENIO (PAC08-0154-9262), and PEGASO/MAGO (TIN2009-13718-C02-01), and by the University of Innsbruck.

References

[1] Cai, Z., X. Yang, and W. Wang, *Business Process Recovery for System Maintenance - An Empirical Approach*, in *ICSM'09*. 2009, IEEE Computer Society. p. 399-402.
[2] Castellanos, M., K.A.d. Medeiros, J. Mendling, B. Weber, and A.J.M.M. Weitjers, *Business Process Intelligence*, in *Handbook of Research on Business Process Modeling*, 2009, Idea Group Inc. p. 456-480.
[3] Dumas, M., W. van der Aalst, and A. Ter Hofstede, *Process-aware information systems: bridging people and software through process technology*. 2005: John Wiley & Sons, Inc.
[4] Günther, C.W. and W.M.P. van der Aalst, *A Generic Import Framework for Process Event Logs.* in *BPI'06*, 2007. LNCS 4103: p. 81-92.
[5] Heuvel, W.-J.v.d., *Aligning Modern Business Processes and Legacy Systems: A Component-Based Perspective (Cooperative Information Systems)*. 2006: The MIT Press.
[6] Ingvaldsen, J.E. and J.A. Gulla, *Preprocessing Support for Large Scale Process Mining of SAP Transactions.* in *BPI'07*. 2008. LNCS 4928: p. 30-41.
[7] Mens, T., *Introduction and Roadmap: History and Challenges of Software Evolution* Software Evolution (Springer Berlin Heidelberg), 2008. 1: p. 1-11.
[8] Newcomb, P., *Architecture-Driven Modernization (ADM)*, in WCRE'05. p. 237.
[9] Pérez-Castillo, R., I. García-Rodríguez de Guzmán, O. Ávila-García, and M. Piattini, *MARBLE: A Modernization Approach for Recovering Business Processes from Legacy Systems*, in *REM'09*. 2009, p. 17-20.
[10] Ratiu, D., *Reverse Engineering Domain Models from Source Code*, in *REM'09*. 2009, p. 13-16.
[11] Ulrich, W.M., *Legacy Systems: Transformation Strategies*. 2002: Prentice Hall. 448.
[12] van der Aalst, W. and A.J.M.M. Weijters, *Process Mining*, in *Process-aware information systems: bridging people and software through process technology*. 2005, John Wiley & Sons, Inc. p. 235-255.
[13] van der Aalst, W.M.P., B.F. van Dongenm, C. Günther, A. Rozinat, H.M.W. Verbeek, and A.J.M.M. Weijters, *ProM : the process mining toolkit*, in *BPM'09*. 2009. p. 1-4.
[14] Weske, M., *Business Process Management: Concepts, Languages, Architectures*. 2007, Leipzig, Alemania: Springer-Verlag Berlin Heidelberg. 368.
[15] Zou, Y. and M. Hung, *An Approach for Extracting Workflows from E-Commerce Applications*, in *ICPC'06*. 2006, IEEE Computer Society. p. 127-136.