Softw Syst Model (2014) 13:1117–1139
DOI 10.1007/s10270-012-0285-5

THEME SECTION PAPER

Assessing event correlation in non-process-aware information systems

Ricardo Pérez-Castillo · Barbara Weber · Ignacio García-Rodríguez de Guzmán · Mario Piattini · Jakob Pinggera

Received: 31 October 2011 / Revised: 19 July 2012 / Accepted: 20 August 2012 / Published online: 18 September 2012

Abstract Many present-day companies carry out a huge amount of daily operations through the use of their information systems without ever having done their own enterprise modeling. Business process mining is a well-proven solution which is used to discover the underlying business process models that are supported by existing information systems. Business process discovery techniques employ event logs as input, which are recorded by process-aware information systems. However, a wide variety of traditional information systems do not have any in-built mechanisms with which to collect events (representing the execution of business activities). Various mechanisms with which to collect events from non-process-aware information systems have been proposed in order to enable the application of process mining techniques to traditional information systems. Unfortunately, since business processes supported by traditional information systems are implicitly defined, correlating events into the appropriate process instance is not trivial. This challenge is known as the event correlation problem. This paper presents an adaptation of an existing event correlation algorithm and incorporates it into a technique in order to collect event logs from the execution of traditional information systems. The technique first instruments the source code to collect events together with some candidate correlation attributes. Based on several well-known design patterns, the technique provides a set of guidelines to support experts when instrumenting the source code. The event correlation algorithm is subsequently applied to the data set of events to discover the best correlation conditions, which are then used to create event logs. The technique has been semi-automated to facilitate its validation through an industrial case study involving a writer management system and a healthcare evaluation system. The study demonstrates that the technique is able to discover an appropriate correlation set and obtain well-formed event logs, thus enabling business process mining techniques to be applied to traditional information systems.

Keywords Business process mining · Event correlation · Event model · Case study

Communicated by Dr. Tony Clark, Balbir Barn, Alan Brown, and Florian Matthes.

R. Pérez-Castillo (✉) · I. G.-R. de Guzmán · M. Piattini
Instituto de Tecnologías y Sistemas de Información (ITSI), University of Castilla-La Mancha, Paseo de la Universidad 4, 13071 Ciudad Real, Spain
e-mail: ricardo.perez.del.castillo@gmail.com; ricardo.pdelcastillo@uclm.es

I. García-Rodríguez de Guzmán
e-mail: ignacio.grodriguez@uclm.es

M. Piattini
e-mail: mario.piattini@uclm.es

B. Weber · J. Pinggera
University of Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria
e-mail: barbara.weber@uibk.ac.at

J. Pinggera
e-mail: jakob.pinggera@uibk.ac.at

1 Introduction

Complex and large organizations are increasingly using Enterprise Modeling technologies to analyze their business units, processes and resources, in addition to the information systems that support these business processes at an operational level. The principal objective of enterprise architecture modeling is to improve the alignment between business processes and information systems by making the impact of planned changes explicit [1].

If the alignment of business processes and information systems is to be achieved then it is first necessary to obtain

an accurate and up-to-date representation of the enterprise architecture itself [2] (e.g., the design and collection of a set of business process models using a modeling language). Unfortunately, these representations cannot be obtained by ignoring enterprise information systems, since these systems implement the majority of organizations' daily operations. What is more, information systems are not a static entity, but are maintained and modernized to cope with new business requirements. Uncontrolled maintenance over time implies that particular business knowledge is embedded in enterprise information systems and is not present anywhere else [3]. This problem makes it necessary to discover and explicitly represent the embedded business knowledge in order to aid enterprise modeling tasks.

Business process mining techniques can be employed to discover and reconstitute the underlying business processes supported by information systems. Business process mining groups well-proven techniques which are used to discover the embedded business processes that are supported by existing information systems [4]. All these techniques employ event logs, which represent the execution of business activities, as a common input. Event logs are often recorded by process-aware information systems (PAIS) [e.g., enterprise resource planning (ERP) or customer relationship management (CRM) systems]. The process-aware nature of PAIS facilitates the direct registration of events during process execution. However, there are a wide variety of non-process-aware information systems (denoted as *traditional information systems* in this paper) that do not have any in-built mechanisms with which to collect events logs. In other cases, when mechanisms exist to record event logs, these logs might not contain sufficient information to discover accurate business processes from traditional information systems.

In order to enable the application of process mining techniques to traditional information systems, a previous work by the authors of this paper presented a technique with which to collect events from such systems [5,6]. This technique first injects statements into the existing source code in order to instrument it. This stage is probably the most complicated since experts have to choose the parts of the source code that must be instrumented or ignored. This approach improves the instrumentation stage by providing a set of guidelines (based on frequently used and well-proven design patterns) to aid the instrumentation of source code. Once the source code has been instrumented, the information system is then able to record (during system execution) certain events that are eventually analyzed and organized in an event log. When applied to real-life traditional systems, the accuracy of the aforementioned technique has been limited owing to the rather simplistic event correlation strategy employed [events were correlated in different process instances (also known as cases) using various simple heuristics] [6,7]. Since it is possible to have various instances of the business process running at the same time, event correlation becomes a key challenge, which consists of the assignment of each event to the correct instance. In addition, owing to the fact that business processes supported by traditional systems are implicitly defined, correlating events into the appropriate execution instance is not trivial.

1.1 Main contributions

The ultimate goal is to support the event log collection from traditional (non-process-aware) information systems by correlating events in the appropriate business process instance. The main contributions of this paper are the following:

1. This paper provides an enhanced technique with which to obtain event logs from traditional information systems, thus addressing the event correlation challenge. Two principal improvements have been made to the preliminary technique presented in [6]:
   (a) The new technique incorporates a set of instrumentation guidelines with which to support experts when selecting the most suitable pieces of source code to be instrumented. These guidelines signify that the instrumentation of insignificant parts of source code, such as auxiliary technical code, is no longer necessary. The candidate correlation data values are then recorded together with each event by means of an instrumented version of a traditional system. In turn, the selective instrumentation of source code reduces noise problems when events are recorded (i.e., events that do not represent business activities will not recorded).
   (b) While the previous technique arbitrarily incorporates the correlation data (i.e., certain data in source code that is then used to correlate process instances), the new technique allows various candidate correlation data to be identified first, and these will then be analyzed to discover the most accurate correlation data set.
2. After recording events during system execution, the technique applies an algorithm to the intermediate information to discover the sub-set of correlation attributes and conditions. The algorithm is adapted from an existing correlation algorithm initially developed for Web service based systems [8]. The correlation set is eventually used to record accurate process instances in a well-formed event log that can be used to discover the embedded business processes.
3. A further key contribution is the empirical validation of the technique with real-life systems. The case study was first executed with two real-life traditional systems: a

healthcare system for the collection of patient reported outcome data and an author management system. To conduct the case study and facilitate the adoption of the technique by industry, the technique has been developed using database-stored intermediate information and a set of algorithms implemented as stored procedures [9]. Second, the obtained results were compared with regards to two previous case studies conducted with the same systems so that the gain of the new approach can be validated.

The main implication of the empirical study is that all the efforts of the business process mining field can be reused and applied to traditional information systems. Enterprise modeling is thus facilitated when it deals with misaligned business processes embedded in existing information systems.

The main benefit of this new approach is that it provides a better event correlation by choosing between certain candidate choices, which involves obtaining more accurate and consistent event logs, and it therefore also mines more accurate business processes.

The remainder of the paper is organized as follows. Section 2 summarizes related work. Section 3 provides a detailed description of the technique proposed to obtain event logs. Section 4 shows the design and execution of the multi-case study. Finally, Sect. 5 discusses conclusions and future work.

2 Related work

Since business process mining techniques consider event logs as input, the collection and management of these logs is a common research topic in literature. For example [10] focus on ERP systems to obtain event logs from the SAP's transaction data logs. In addition [11] provide a generic import framework with which to obtain event logs from different kinds of PAIS, and deals with some interesting challenges such as event correlation. While these proposals focus on PAIS, our proposal addresses the collection of event logs from traditional information systems that do not have any in-built mechanisms with which to record events.

Moreover, event correlation is an issue of growing importance within the process mining field. Events can usually be correlated in different ways. Most authors therefore consider event correlation to be a subjective activity, and most proposals in literature attempt to correlate events using heuristics [12]. Most correlation event techniques assess certain indicators and check whether they are under or over a heuristic threshold in order to prune non-promising correlation attributes and conditions. In this respect, Burattin et al. [13] focus on event data sets with an absence of correlation attributes and propose an approach which introduces a set of extra fields, decorating each single activity collected as an event in the log which is known to carry the information about the process instance. Algorithms are designed using relational algebraic notions in order to extract the most promising case ids from the extra fields.

Rozsnyai et al. [14] propose algorithms with which to discover correlation rules using statistical indicator (e.g., variance of attribute values) assessments from data sets. Ferreira et al. [15] similarly propose a probabilistic approach with which to discover the case id in unlabeled event logs.

Other techniques such as the algorithms proposed by Kato et al. [16] correlate events by directly analyzing the source code. This technique attempts to discover the common working package for a set of events recorded after the execution of a particular piece of source code.

Moreover, Motahari-Nezhad et al. [8] propose a set of algorithms with which to discover the necessary correlation attributes and conditions (e.g., conjunctive and disjunctive conditions grouping two or more correlation attributes) from the available attributes of Web service interaction logs. This paper improves on a previous technique used to retrieve event logs from traditional systems [6], which employs certain algorithms to discover the correlation set as suggested by Motahari-Nezhad et al. [8]. While the algorithm of Motahari-Nezhad et al. is applied to Web service logs, the approach presented herein adapts the algorithm to be applied to traditional information systems in order to obtain event logs.

Event correlation becomes a non-trivial task as a consequence of the increasing heterogeneity and distribution of enterprise information systems. Some proposals address the distribution of heterogeneous event logs. For example, Myers et al. [17] apply generic distributed techniques in conjunction with existing log monitoring methodologies in order to obtain additional insights into event correlation. In addition, Hammoud [18] presents a decentralized event correlation architecture. Decentralized event correlation approaches are not, however, within the scope of this paper.

3 Technique used to obtain event logs

The challenge of the technique presented in this paper lies in the generation of event logs from traditional information systems by paying special attention to the correlation of the event. Despite the existence of technologies with which to collect execution traces from java virtual machine (JVM) and other automatic logging mechanisms, this technique relies on the instrumentation of source code with the aid of experts since it filters out non-relevant information at runtime.

Figure 1 provides an overview of the technique proposed to obtain event logs from traditional information systems, which consists of four main stages:

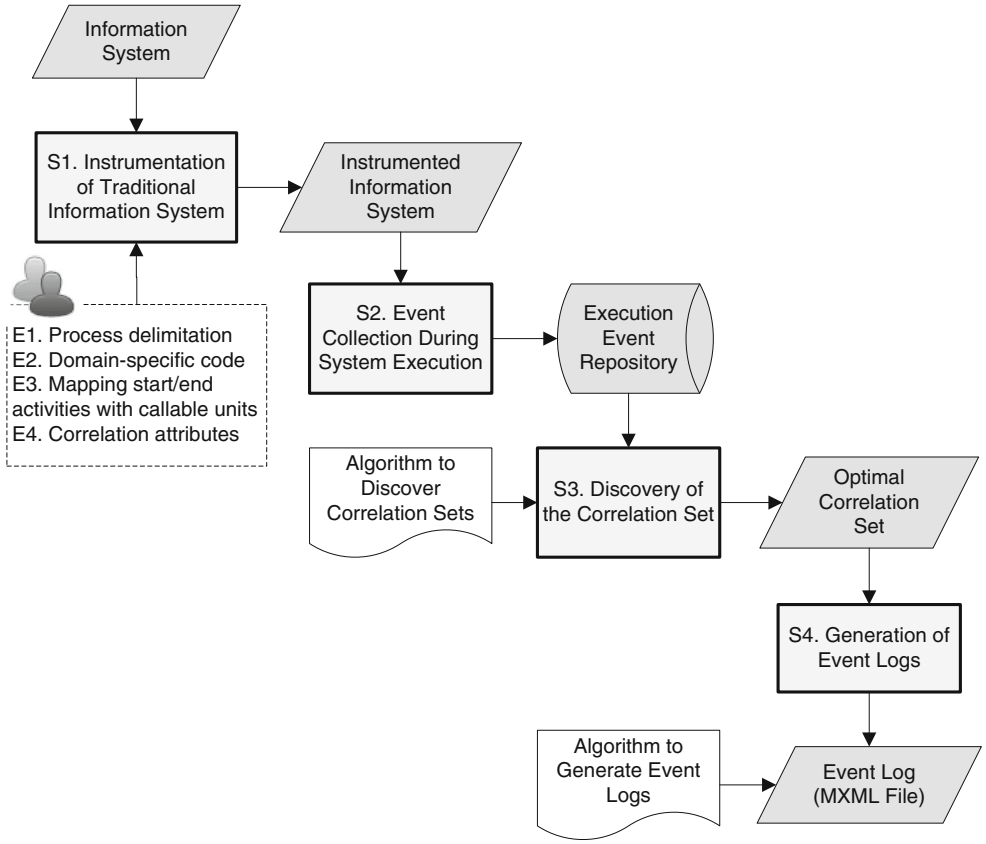


Fig. 1 An overview of the technique proposed to obtain event logs

1. The objective of the first stage is to instrument traditional systems so that they can collect events during their execution. Experts identify certain information such as candidate correlation attributes, whose runtime values will then be collected together with each event (cf. Sect. 3.1). This approach extends the previous technique with a set of guidelines for business experts in order to facilitate the instrumentation of source code.
2. In the second stage, the modified system is executed and events are progressively recorded. As a result, events and their respective attributes are then stored in a database in an intermediate format (cf. Sect. 3.2).
3. The third stage applies an adaptation of the algorithm proposed by [8] to the event data sets in order to discover the set of attributes and conditions needed to correlate events (cf. Sect. 3.3).
4. Finally, the last stage applies an algorithm by considering the correlation set in order to correlate each event with its corresponding process instance (cf. Sect. 3.4). A standard-format event log is therefore obtained from the traditional system.

3.1 Instrumentation of traditional information systems

Since traditional information systems do not have any built-in mechanisms with which to record events concerning the business processes executed, this stage attempts to instrument thus enabling them to record events (see Fig. 1).

This stage is semi-automatic. A parser syntactically analyzes the source code (statement by statement) and automatically injects statements into particular places in the code in order to collect events during system execution. Statements are injected into callable units (pieces of source code that can be invoked, e.g., Java methods, C procedures or Visual Basic functions, among others). The tool used to inject statements can easily be extended with parsers in order to support different programming languages since it follows a plugin-oriented architecture.

This work follows the 'a callable unit/a business activity' approach proposed by Zou et al. [19], i.e., callable units are the generic elements into which the parser injects statements to record an event corresponding to the execution of a business activity. However, not all the executions of

Table 1 Steps used to instrument source code

Id	Step	Deliverables	Aided by
E1	Delimit processes	Process names, in addition to their initial and end business activities	Documentation of organization
E2	Indicate business domain code to be instrumented	Set of source code files and packages	Experts' guidelines
E3	Map start/end activities with callable units	Pairs of start/end business activities selected in E1 and callable units (i.e., java methods)	System analysts
E4	Provide candidate correlation attributes	Set of classifiers (i.e., Java classes or interfaces)	List of all possible classifiers belonging to the code selected in E2

callable units have to be recorded as events. Some callable units such as fine-grained or technical callable units do not correspond to events and must be discarded. Injection into the appropriate place is consequently aided by information provided by experts (see Fig. 1). These experts, who have business process and information system skills, identify the following information through the use of four steps (see Table 1).

3.1.1 Delimiting business processes

Firstly, business experts delimit business processes in Step E1. This delimitation is carried out by providing the name of tentative business processes in addition to the start and end activities of these processes. This information is necessary owing to the fact that the definition of business processes in non-process-aware information systems is not implicit, since these systems are oriented towards procedures or functions rather than processes [5]. This information is then used to associate additional information during the following steps, which provide the source code to be instrumented (Step E2), a mapping between the start/end activities and the pieces of source code that support them (Step E3).

3.1.2 Choosing domain code to be instrumented

The source code of traditional information systems can be classified as either (1) problem domain code (also known as business or domain code) which supports the business activities of the underlying business processes or (2) solution domain code (also known as technical code) which supports auxiliary code for the first kind of code. In order to reduce potential noise in the event log, which is caused by technical source code (i.e., auxiliary code that does not support any business rule in the system), the business experts also examine the legacy source code and select those directories, files or sets of callable units that support business activities (i.e., they select the callable units belonging to the problem domain) (see E2 in Table 1).

This selection can be quite challenging for business experts since documentation is often missing or misleading. Even when a clear and rich documentation is available, experts need to acquire the best program comprehension to provide a good selection of the domain code. We have dealt with this challenge by considerably extending the previous technique through the development of a set of guidelines to support this endeavor. In order to effectively select the code to be instrumented, business experts are advised to discard, in a first step, whole code packages (e.g., by considering the architectural characteristics and design patterns used in the system) and to then, in a next step, analyze the remaining packages. This second step is typically required, since code packages do not always clearly separate business code from technical code (we refer to this as *delocalization* and the *interleaving* challenge [20]).

Figure 2 shows a characteristic architecture of an object-oriented system design which is organized in three layers and includes a set of well-known design patterns. Class names represent the role each class plays in a certain design pattern. The classes highlighted in Fig. 2 represent the kind of classes that should eventually be instrumented according to the following guidelines. These guidelines are discussed and illustrated using the two case studies described in Sect. 4 [i.e., computer-based health evaluation system (CHES), a healthcare system for the collection of patient reported outcome data; and AELG-Members, an author management system].

G1. Check for three-tier architecture

When selecting the code packages to be instrumented it may be helpful to consider the architectural characteristics of the system under investigation. Many system designs follow the three-tier design pattern, which consists of the decomposition of a system architecture into three layers [21]: (1) the *domain* layer, which supports all the business entities and controllers, and can thus be considered as a business domain code; (2) the *presentation* layer, which deals with the user interfaces; and (3) the *persistency* layer, which handles the data access. For the instrumentation, only the domain layer is

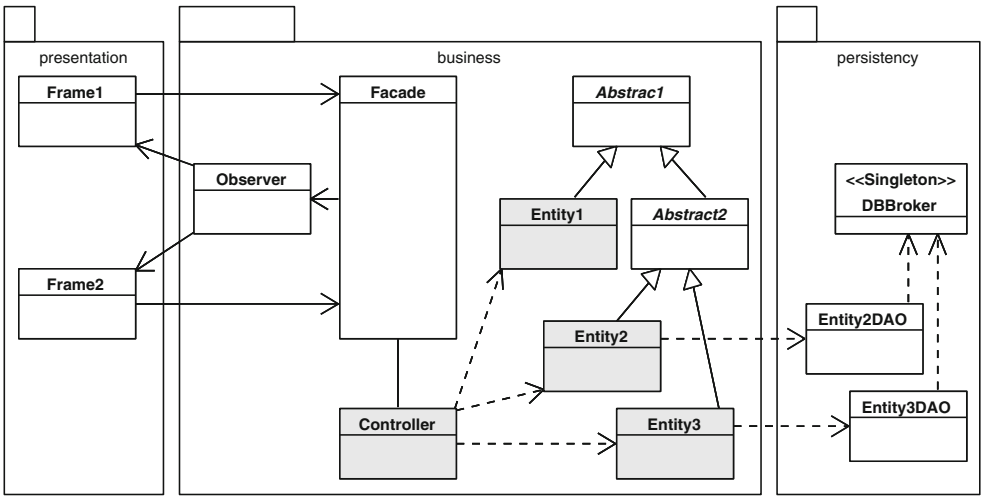


Fig. 2 Frequent system design following the three-tier architecture and other design patterns

relevant, whereas the presentation layer and the persistency layer typically comprise auxiliary code. The business expert can exploit the fact that in many cases the different layers are organized into separate code packages and can thus be relatively easily identified and discarded from instrumentation.

For example, both systems under study have at least these layers. CHES is an extensible systems based on a set of plug-ins. However, CHES has some plug-ins that are in charge of the graphical user interface and others that are in charge of the management of the business entities. The AELG-Members' design is structured in four packages: *model*, *view*, *controller* and *util*. The *model* and *controller* packages correspond to the business layer, the *view* package is related to the presentation and the *util* package is associated with persistency and other auxiliary operations.

After discarding whole code packages based on the application's architecture, e.g., three-tier architecture, business experts can identify further classes to be discarded based on the usage of other design patterns. We explicitly state guidelines for various design patterns here. Similar guidelines can be derived for other design patterns. Design patterns are often applied in order to easily and quickly separate business code and technical code in a non-coupling manner, and they can thus serve as heuristics with which to instrument code.

G2. Check for observer classes

Most system designs are optimized in order to reduce the code coupling between packages or classes [21]. Many designs therefore implement the observer pattern [22], which defines a one-to-many dependency between objects so that when one object changes state, all its dependents are automatically notified and updated. This pattern is often used to reduce coupling between the presentation and domain layer since all the different views in the user interface are notified by the observer class when the domain classes change their state. It is suggested that domain experts should seek the occurrence of the observer pattern in the code in order to check whether the respective classes can be discarded. If observers are utilized to update the user interface, observer classes can safely be discarded. On the contrary, if observers perform operations on domain code, business experts should check whether the domain code should be instrumented and observers can be discarded.

CHES implements the observer pattern since it provides different plug-ins which are in charge of the presentation and the business domain, which are not coupled. This signifies that a different presentation plug-in could be used in the CHES. In the case of CHES, the classes that implement the observer pattern do not contain the business domain code to be instrumented and could be discarded.

G3. Check for data-access object (DAO) pattern

Another pattern that can frequently be found is the DAO pattern [23]. This design pattern provides a technique with which to separate object persistence and data access logic from any particular persistence mechanism. The DAO approach provides the flexibility needed to change an application's persistence mechanism over time without the need to re-engineer application logic which interacts with the persistency layer. DAO classes are often used in the persistency layer to stores/retrieve a snapshot of the internal state of an object when it becomes persistent in a database. DAO classes therefore constitute potential candidate classes for being discarded from the instrumentation phase.

AELG-Members contains a DAO class in the model package (the business layer) for each business entity class. For example, the *AuthorDAO* class is associated with the *Author* class, the *MemberDAO* class is associated with the *Member*

class, and so on (see Fig. 2). In this case, all the DAO classes are discarded in order to permit instrumentation.

G4. Check for object factories

Code packages that represent business code often have many classes which represent business entities (e.g., customer, account, invoice, product, etc.). These entities are sometimes instantiated by means of creational patterns such as the factory method or abstract factory. These patterns have in common the usage of abstract classes in hierarchical structures, together with concrete leaf classes with which to instantiate different business domain objects [22]. Owing to the fact that abstract classes cannot be instantiated (and the instrumentation will not therefore have any effect) only the leaf classes in the inheritance tree of these patterns should be considered as the entities to be instrumented (see highlighted classes in Fig. 2).

AELG-Members implements various abstract factories. For example, this system defines an abstract factory with which to create different kinds of queries within the *aelg.model.query* package. In this case, leaf classes (i.e., *QueryBooleanTable*, *QueryMultiValueTable, etc.*) should be instrumented while the abstract classes like *QueryElement* is are discarded.

G5. Check for singleton classes

Many system designs apply the singleton pattern [22], which ensures that a class has only one instance, and provides a global point of access to it. This is often used with technical classes that provide a clear and independent auxiliary functionality (e.g., a database access manager). If the singleton pattern is used, then these classes should be considered as technical code and discarded.

Both CHES and AELG-Members contain several singleton classes. For instance, in the case of AELG-Members there is a clear example—the *DateUtil* class, which is in charge of managing dates.

G6. Check for facade classes

Many systems provide a unified access to a set of functionalities or services in a subsystem which are known as facades [22]. A facade defines a higher-level interface that makes the subsystem easier to use. Facades do not contain additional business domain code, and can therefore be discarded (see Fig. 2). However, they might call operations in business domain code that have to be instrumented. If a facade class is detected, it is necessary to check whether operations in domain classes are invoked by the facade and discard the facade.

AELG-Members contains three different facades within the business layer (i.e., in the model package): *AuthorsFacade*, *MembersFacade* and *CategoriesFacade*. These classes respectively collect the set of available actions that can be carried out with the systems concerning authors, members of the organization and author categories. The facade classes were discarded during the instrumentation of source code since the end-point methods (which are called from the facade) are available in others classes of the business domain layer.

These guidelines entail a set of heuristics to help business experts. However, this set of guidelines has some limitations. For example, in our particular case, the guidelines are useful for object-oriented systems following a tier-based architecture. However, the initial set of guidelines could be extended with more guidelines based on different patterns, or even adapted to other kinds of architectures or platforms. Nevertheless, we believe that an instrumentation aided by a set of guidelines based on design patterns provides better results than instrumentation without this kind of guidelines.

3.1.3 Mapping start and end activities

Thirdly, experts also provide a mapping between the set of start and end business activities and the callable units supporting them (see E3 in Table 1). This information is needed to know which callable units are associated with the start or end points of the underlying business process that business experts had previously established in step 1.

3.1.4 Identifying candidate correlation attributes

Finally, experts identify those code elements that can be treated as candidate correlation attributes (see E4 in Table 1). This stage is supported by another improvement to the tool presented in [6], which additionally supports the identification and addition of candidate correlation attributes. These candidate attributes will be used to correlate events (cf. Sect. 3.3). The selection of candidate correlation attributes is the most important task in this stage, since an incomplete list of candidate attributes may lead to a non-suitable correlation. The tool provides experts with all the possible selectable attributes: (1) all the parameters that appear in callable units and (2) the output and fine-grained callable units that are invoked within those callable units that are considered to be collected as events. Experts then select a subset of candidate correlation attributes. During the static analysis of source code, the information concerning candidate correlation, and other information provided by experts is automatically injected with tracing statements in callable units, thus enabling event collection.

Figure 3 provides an example of the results obtained after instrumenting a Java method of one of the systems under study (cf. Sect. 4). The two *tracing* statements (see highlighted statements) are injected at the beginning and at the end of the body of the method. Those candidate correlation

```
public class SocioFacadeDelegate {
    [...]
    public static void saveAuthor(AuthorVO author) throws InternalErrorException {
        writeDBEvent("SocioFacadeDelegate.saveAuthor", "Author Management", "", "start",
          false, false, -1, false, 2, 8, "", "" + author.getId(), "" + author.isHistorico(),
          "" + author.getNumeroSocio(), "", "" + author.getCotas());
        try {
            SaveAuthorAction action = new SaveAuthorAction(author);
            PlainActionProcessor.process(getPrivateDataSource(), action);
        } catch (InternalErrorException e) {
           throw e;
        } catch (Exception e) {
           throw new InternalErrorException(e);
        }
        writeDBEvent("SocioFacadeDelegate.saveAuthor", "Author Management", "", "complete",
          false, true, -1, false, 2, 8, "", "" + author.getId(), "" + author.isHistorico(),
          "" + author.getNumeroSocio(), "", "" + author.getCotas());
    }
    [...]
}
```

Fig. 3 Example of code instrumentation for a method of an author management system

attributes that are present in a method (e.g., a parameter or variable) are automatically injected into the *tracing* statements, i.e., the respective variables are in the set of parameters of the invocation to the method *'writeDBEvent'* (see Fig. 3). However, not all correlation attributes defined by experts are present in all methods (e.g., owing to the absence of a particular variable). In this case, the respective parameter of the method *'writeDBEvent'* (i.e., the tracing statement) is an empty string (" "). As a result, during the execution of this method, the runtime value (or an empty value) will be recorded together with the name of the attribute and the event (the name of the method representing the business activity).

Despite the fact that the instrumentation stage is semi-automatic and aided by business experts' opinion, the technique (through its respective supporting tool) is scalable for both larger business processes and a large number of different processes supported by a system. This is owing to the fact that the time taken to statically analyze larger information systems would be greater, while the time taken by experts to provide information to aid in the instrumentation would often be the same, since it does not depend on the size of business processes.

3.2 Collection of events

The event collection stage is in charge of the appropriate generation and storage of events throughout system execution. The instrumented system is executed and (when an injected statement is executed) records events together with the value of all the candidate correlation attributes available in that callable unit (see Fig. 1). The new technique has a key difference as regards the preliminary technique. While other similar techniques [6,7] build an event log on the fly (i.e., when an event occurs it is directly recorded in the event log), this new technique stores all the information about events and their candidate correlation attributes in an intermediate database for later analysis. Another important difference is that the new technique collects various candidate correlation attributes whilst previous techniques collect only the correlation attribute selected by experts during the instrumentation stage.

This technique uses a relational database context in a similar way to other techniques such as [8]. Relational databases, in comparison to other options like XML files, facilitate the implementation of algorithms to discover the correlation set

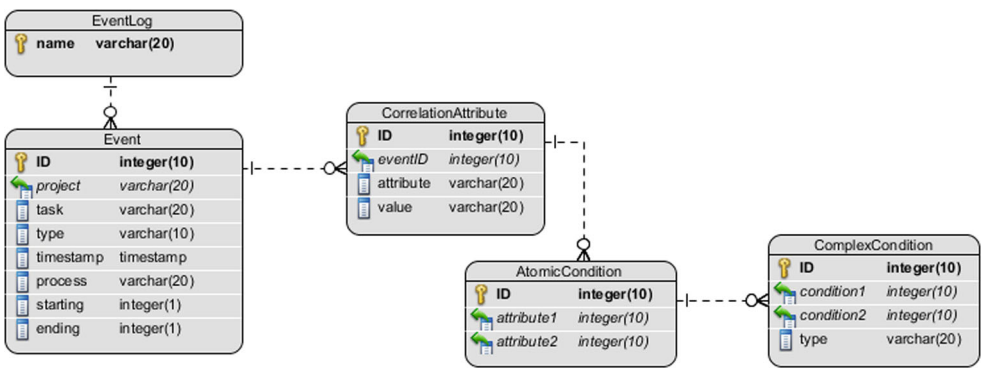
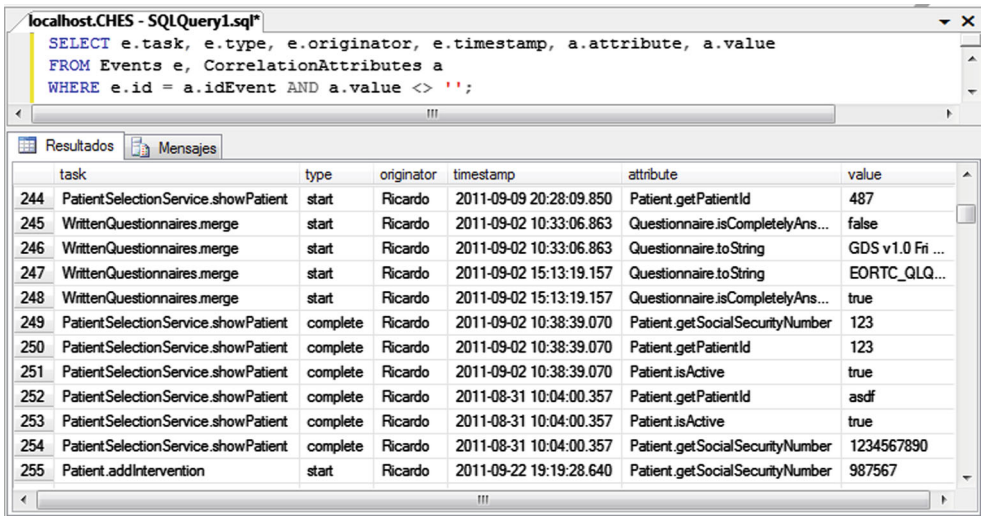


Fig. 4 Database schema for event and correlation attribute collection



localhost.CHES - SQLQuery1.sql*

```
SELECT e.task, e.type, e.originator, e.timestamp, a.attribute, a.value
FROM Events e, CorrelationAttributes a
WHERE e.id = a.idEvent AND a.value <> '';
```

Resultados | Mensajes

	task	type	originator	timestamp	attribute	value
244	PatientSelectionService.showPatient	start	Ricardo	2011-09-09 20:28:09.850	Patient.getPatientId	487
245	WrittenQuestionnaires.merge	start	Ricardo	2011-09-02 10:33:06.863	Questionnaire.isCompletelyAns...	false
246	WrittenQuestionnaires.merge	start	Ricardo	2011-09-02 10:33:06.863	Questionnaire.toString	GDS v1.0 Fri ...
247	WrittenQuestionnaires.merge	start	Ricardo	2011-09-02 15:13:19.157	Questionnaire.toString	EORTC_QLQ...
248	WrittenQuestionnaires.merge	start	Ricardo	2011-09-02 15:13:19.157	Questionnaire.isCompletelyAns...	true
249	PatientSelectionService.showPatient	complete	Ricardo	2011-09-02 10:38:39.070	Patient.getSocialSecurityNumber	123
250	PatientSelectionService.showPatient	complete	Ricardo	2011-09-02 10:38:39.070	Patient.getPatientId	123
251	PatientSelectionService.showPatient	complete	Ricardo	2011-09-02 10:38:39.070	Patient.isActive	true
252	PatientSelectionService.showPatient	complete	Ricardo	2011-08-31 10:04:00.357	Patient.getPatientId	asdf
253	PatientSelectionService.showPatient	complete	Ricardo	2011-08-31 10:04:00.357	Patient.isActive	true
254	PatientSelectionService.showPatient	complete	Ricardo	2011-08-31 10:04:00.357	Patient.getSocialSecurityNumber	1234567890
255	Patient.addIntervention	start	Ricardo	2011-09-22 19:19:28.640	Patient.getSocialSecurityNumber	987567

Fig. 5 Data set example obtained from CHES, one of the systems used in the assessment

from a vast amount of data with a good performance (cf. Sect. 3.3).

Figure 4 shows the relational database schema used to represent the intermediate event information. The *EventLog* table is used to represent different logs obtained from different source systems. The *Event* table contains all the different events collected, including the business task executed, type (start or complete), originator, execution timestamp, two columns to indicate whether the executed task is the initial or final task in a process, and the process name. *CorrelationAttribute* is a table related to the *Event* table and contains the runtime values of candidate correlation attributes together with the reference to the event for which the correlation attribute was collected.

Candidate correlation attributes are combined by means of correlation conditions which are then used to correlate events. According to the categorization of Motahari–Nezhad et al. [8], two kinds of correlation conditions are differentiated: atomic conditions and complex:

1. *Atomic conditions* represent key-based conditions which compare two correlation attributes. For instance, *condition1:attribute1=attribute2* signifies that two events will be correlated if the value of *attribute1* of the first event is equal to the value of *attribute2* of the second event under evaluation. These conditions are stored in the *AtomicCondition* table (see Fig. 4).
2. *Complex conditions* evaluate the simultaneous aggregation of two different conditions, which are combined by a logic operator, e.g., conjunction (*AND*) or disjunction (*OR*). For example, *condition3: condition1 AND condition2* evaluated for two events signifies that both atomic conditions (*condition1* and *condition2*) must be simultaneously met for the two events. The *Complex-Condition* table represents this information in the database schema (see Fig. 4).

Figure 5 shows, as an example, a view of the data collected after source code instrumentation. It contains the aforementioned data: task, type, originator and timestamp for each event recorded, in addition to correlation attributes and their values.

3.3 Discovering the event correlation set

After the collection of events and candidate correlation attributes, the intention of the third stage is to discover the event correlation set. Event correlation deals with the definition of relationships between two or more events in order to discover which events belong to the same business process execution (i.e., process instance). Event correlation is important in the identification of business processes in traditional information systems, since the business processes executed are not explicitly defined through the existing source code [5].

Figure 6 shows an overview of the event correlation challenge. Each business process can be executed several times, and even in parallel. Each execution is known as a process instance. All the events recorded during system execution must in turn be correlated into the correct process instance. For example, two different events may refer to the execution of the same business activity (e.g., *receive invoice*). However, they may belong to two different instances (e.g., two instances that are executed for two different customers).

An adaptation of the algorithm described in [8] is applied to discover the correlation set (see Algorithm 1). A correlation set consists of (1) a set of atomic conditions (i.e., equal comparisons between pairs of correlation attributes) and (2) a set of complex conditions (i.e., conjunctive com-

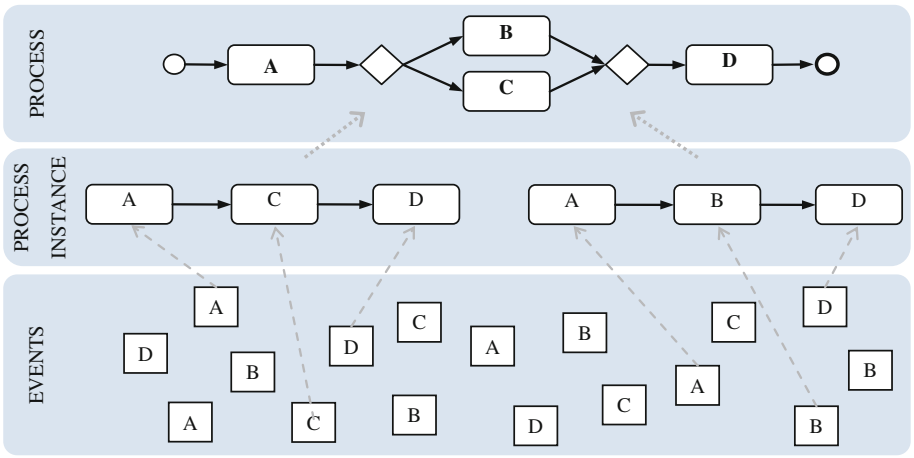


Fig. 6 Event correlation challenge

parisons between pairs of atomic conditions). Unlike [8], this technique does not consider disjunctive conditions since these conditions are only needed for heterogeneous systems to detect synonyms of certain correlation attributes [8], and these kinds of systems are not within the scope of this paper.

3.3.1 Discovery of atomic conditions

Algorithm 1 first considers every possible combination of two candidate correlation attributes which may be involved in atomic conditions (lines 1–3). The algorithm then prunes the non-interesting conditions (i.e., the less promising conditions) using the following three rules and paying attention to different criteria.

Algorithm 1. Discovery of the correlation set

Input: *Attributes*; *Events*
Output: *AC*: the set of atomic conditions; *CC*: the set of conjunctive conditions
1: **for** a ∈ *Attributes* ∧ b ∈ *Attributes* **do**
2: *AC* ← "a = b"
3: **end for**
4: *AC* ← *AC* – { c | c.attribute1 = c.attribute2 **and** (*DistinctRatio* (c.attribute1) < α **or** *DistinctRatio* (c.attribute1) = 1) }
5: *AC* ← *AC* – { c | c.attribute1 ≠ c.attribute2 **and** *SharedRatio* (c.attribute1, c.attribute2) < α }
6: *AC* ← *AC* – { c | *PIRatio* (c.attribute1, c.attribute2) < α **or** *PIRatio* (c.attribute1, c.attribute2) > β }
7: N_0 ← *AC*; N_1 ← { }
8: k ← 1
9: **for** c1 ∈ N_{k-1} **and** c2 ∈ N_{k-1} **do**
10: N_k ← "c1 ∧ c2"
11: **end for**
12: **while** N_k ≠ { } **do**
13: N_k ← N_k – { c | *ConjNumberPI* (N_k.condition1, N_k.condition2) ≤ *NumberPI* (N_k.condition1.attribute1, N_k.condition1.attribute2) **or** *ConjNumberPI* (N_k.condition1, N_k.condition2) ≤ *NumberPI* (N_k.condition2.attribute1, N_k.condition2.attribute2) }
14: N_k ← N_k – { c | *ConjPIRatio* (N_k.condition1, N_k.condition2)<α **or** *ConjPIRatio* (N_k.condition1, N_k.condition2) > β }
15: *CC* ← *CC* ∪ N_k
16: **for** c1 ∈ N_k **and** c2 ∈ N_k **do**
17: N_{k+1} ← "c1 ∧ c2"
18: **end for**
19: k ← k+1
20: **end while**

Rule 1. Pruning using distinct ratio of attributes

The first rule attempts to prune atomic conditions according to the different values of the candidate correlation attributes. For example, let us imagine the candidate attribute *isPremium,* which is defined about customers and probably has two possible values, true or false. If this attribute is used as a correlation attribute, the technique would split the data set of events into only two process instances for premium and non-premium customers, which would not be a good correlation.

The first rule prunes candidate atomic conditions by evaluating the *DistinctRatio* (Eq. 1), which indicates the cardinality (i.e., the number of different values) of an attribute in the data set regarding its total number of non-null values.

When attributes of atomic conditions are the same, the distinct ratio must be above the alpha threshold or distinct to 1 (see line 4 of Algorithm 1). *Alpha* (Eq. 2) quantifies the variance of values of every attribute regarding the size of the event data set. Alpha is consequently used as the threshold with which to detect global unique values. When the *DistinctRatio* of an attribute is below alpha or one, this signifies that the attribute contains a global unique value and the atomic condition with this attribute can therefore be pruned since this attribute cannot be used to correlate events in two or more process instances.

$$DistinctRatio(a_i) = \frac{distinct(a_i)}{nonNull(a_i)} \quad (1)$$

$$\alpha = \frac{distinct_{MAX}(a_i)}{NumberOfEvents} \quad (2)$$

Rule 2. Pruning using the shared ratio of two different attributes

Rule 2 is similar to Rule 1 but works with two different correlation attributes rather than with atomic conditions involving

the same attribute. Rule 2 uses *SharedRatio* (Eq. 3) rather than *DistinctRatio* to represent the number of distinct shared values (regarding their non-null values) for two different attributes. The atomic condition formed from two different attributes will be pruned when the Shared Ratio of both attributes is above the alpha threshold (see line 5 of Algorithm 1). This rule also discards atomic conditions that correlate all the events in a single process instance.

$$SharedRatio(a_i, a_j) = \frac{distinct(a_i, a_j)}{\max(distinct(a_i), distinct(a_j))} \quad (3)$$

Rule 3. Pruning using the process instance ratio

Atomic conditions are pruned according to the process instance ratio (Eq. 4). This rule checks that the partitioning of the future event log does not only have one or two long instances, or many short instances. *PIRatio* (Eq. 4) is measured as the estimated number of process instances (*NumberPI*) (Eq. 5) divided into non-null values for both attributes.

$$PIRatio(a_i, a_j) = \frac{|NumberPI(a_i, a_j)|}{nonNull(a_i, a_j)} \quad (4)$$

$$\begin{aligned} NumberPI(a_i, a_j) = \{v : \exists e, e' \in Events | \\ e.attribute1 = a_i \; e'.attribute2 = a_j \\ v = e.attribute1.value = e'. \\ attribute2.value \\ e.timestamp > e'.timestamp\} \end{aligned} \quad (5)$$

NumberPI (Eq. 5) is in turn heuristically assessed as the distinct attribute values for all the different couples of events (executed in a row) containing both attributes. The algorithm proposed by Motahari-Nezhad et al. [8] first calculates a set of correlated event pairs, and then suggests computing the number of process instances as the recursive closure over the set of these event pairs. In contrast to that algorithm, our approach estimates *NumberPI* by considering the number of possible pairs of correlated events. This change has been made since the recursive closure evaluation is time-consuming [the complexity of graph closure algorithms in literature is often $O(2^n)$ since they check each pair of nodes for the remaining pairs]. On the contrary, the results expected when using this proposal can be considered as a heuristic approximation with a lower computational cost (i.e., O(n) since this technique only evaluates the list of event pairs). This is a key difference as regards the algorithm proposed by Motahari-Nezhad et al. [8].

Atomic conditions will be discarded when the *PIRatio* value is below alpha or above beta (line 6 of Algorithm 1). *Beta* (Eq. 6) is another of the thresholds used to evaluate the average length of the outgoing instances. For instance, a beta value of 0.5 (a value commonly used) signifies that conditions leading to process instances with a length above or equal to half the total events would be discarded. Higher beta values allow fine-grained process instances to be obtained, while lower values provide larger process instances. The beta threshold is often established between 0.25 and 1, since a beta threshold that is equal or close to 0 may correlate most events in a single, larger process instance.

In practice, the final beta value as provided by business experts is established at the end of an iterative process during which experts can test different beta values. Correlation sets obtained with different beta values only have small differences. In fact, the beta value only represents the degree of restriction regarding the incorporation (or otherwise) of additional correlation attributes into the chosen conditions.

$$\beta \epsilon [0.25, 1] \quad (6)$$

3.3.2 Discovery of conjunctive conditions

After atomic conditions have been filtered out, the algorithm (see Algorithm 1) builds all the possible conjunctive conditions based on the combination of outgoing atomic conditions (lines 7–11). These conditions are then pruned by applying Rules 4 and 5 (see lines 13–14 in Algorithm 1). New conjunctive conditions are then iteratively evaluated by combining the remaining previous conditions (lines 15–19).

Rule 4. Pruning using the monotonicity

The first heuristic (line 13) applied to filter out conjunctive conditions is based on the monotonicity of the number of process instances. This heuristic is based on the idea that the number of process instances for a conjunctive condition is always higher than (or at least equal to) the number for their atomic conditions in isolation. This assumption is often true, since complex conditions are more restrictive, unless complex conditions are subsumed in one of their atomic conditions. Conjunctive conditions that do not increase the number of process instances are therefore pruned. The number of process instances obtained through conjunctive conditions is evaluated as *ConjNumberPI* (Eq. 7) and is based on (Eq. 5), which is defined for simple conditions, and is measured by intersecting both component conditions of the conjunctive condition.

$$\begin{aligned} ConjNumberPI(c_i, c_j) = NumberPI(c_i.a_1, c_i.a_2) \\ \cap \; NumberPI(c_j.a_1, c_j.a_2) \end{aligned} \quad (7)$$

This rule entails another important difference with regard to the algorithm presented in [8], since this algorithm con-

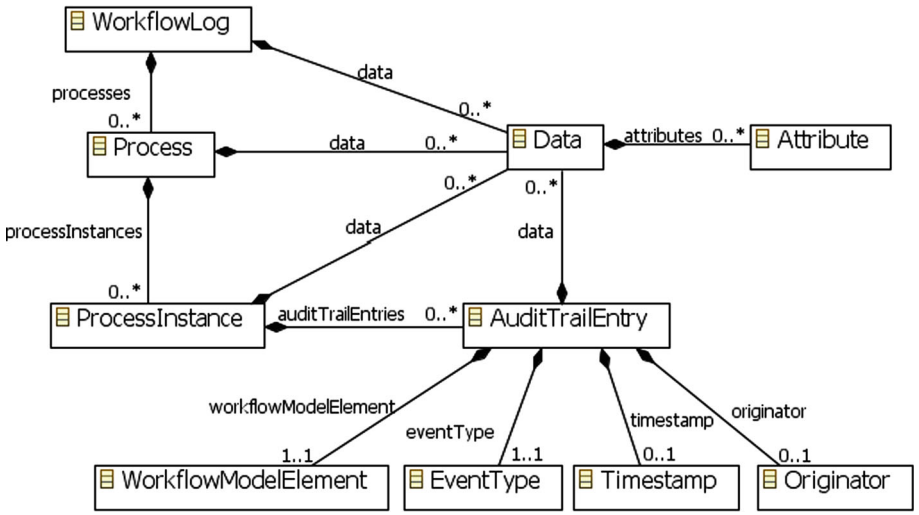


Fig. 7 MXML metamodel to represent event logs

siders the number of process instances (but not the length of instances) to evaluate the monotonicity.

Rule 5. Pruning using the conjunctive process instance ratio

The algorithm also applies the same rules concerning the partitioning of the log to the conjunctive conditions (see line 14 in Algorithm 1). The ratio of process instances is thus also evaluated for conjunctive conditions (*ConjPIRatio*) (Eq. 7). When *ConjPIRatio* is below the alpha or above the beta threshold the conjunctive condition is discarded.

In conclusion, the proposed algorithm adapts the algorithms provided by Motahari-Nezhad et al. [8] in order to adjust it to traditional information systems. There are two main changes, as seen above in this section: (1) the way in which the expected number of process instances is calculated for each condition; and (2) the monotonicity heuristic which only takes into account the estimated number of process instances.

3.4 Generating event logs

After obtaining the correlation set, the fourth stage discovers process instances using the correlation set obtained in order to build the final event log, which will be written following the Mining XML (MXML) format [24].

MXML is a notation based on XML and is the most common format used by most process mining tools [24]. Another interesting standard is extensible event stream (XES) [25], which is also used to record event logs in the same way as MXML. However, while MXML particularly focuses on process mining scenarios, XES provide a generally acknowledged format to support general data mining, text mining, and statistical analysis by prioritizing the extensibility and variability of logs rather than providing a well-known format supported by most common mining tools. This approach uses MXML because of its degree of maturity and the absence of extensibility needs in the main research goal.

The MXML format, as presented in the MXML metamodel shown in Fig. 7, represents an event log as a set of *Process* elements that contain several *ProcessInstance* elements, each of which has a sequence of *AuditTrailEntry* elements (see Fig. 7). Each *AuditTrailEntry* element represents an event and consists of four main elements: (1) the *WorkflowModelElement* that represents the activity executed; (2) the *EventType* showing whether the activity is being executed (start) or had been completed (complete); (3) the *Originator* identifying the user who started or completed the activity; and (4) the *Timestamp* recording the date and time of the event. All of these elements may also have a *Data* element that consists of a set of *Attributes* including related information.

In order to obtain the final MXML event logs, Algorithm 2 correlates all the events of the intermediate data set in its process instance within the event log. The algorithm explores all the candidate events pairs, i.e., those pairs that belong to the same process and were executed in a row (line 2). When an event was recorded as the start point of a process, the target process takes this name (lines 3–5). For each candidate event pair, all the atomic and conjunctive conditions of the correlation set are evaluated (line 7). If the event pair meets all the conditions, then it is a correlated event pair and these events are then put into the correct instance, and the instance is in turn added to the process (lines 9–10). Process instances are previously identified by means of the specific values of the events' attributes involved in the correlation set (line 8). Each process found during the event pair exploration is eventually added to the event log (line 13) together with all the process instances discovered.

Algorithm 2. Discovery of process instances

```
Input: Events; AC: the set of atomic conditions; CC: the set of conjunctive conditions
Output: Log: the final event log
1:  process ← { }; case ← { }
2:  for e1 ∈ Events and e2 ∈ Events and e1.proces = e2.process
    and e1.timestamp ≤ e2.timestamp do
3:      if e1.starting = true and
            ∀ n ∈ Log.processes.name, process.name = n then
4:          process.name ← e1.process
5:      end if
6:      for ac ∈ AC and cc ∈ CC do
7:          if ∃ i, e1.attributes[ i ].name = ac.attribute1 and
            ∃ i', e2.attributes[ i' ].name = ac.attribute2 and
            e1.attributes[ i ].value = e2.attributes[ i' ].value and
            ∃ j, e1.attributes[ j ].name = cc.condition1.attribute1 and
            ∃ j', e2.attributes[ j' ].name = cc.condition1.attribute2 and
            e1.attributes[ j ].value = e2.attributes[ j' ].value and
            ∃ k, e1.attributes[ k ].name = cc.condition2.attribute1 and
            ∃ k', e2.attributes[ k' ].name = cc.condition2.attribute2 and
            e1.attributes[ k ].value = e2.attributes[ k' ].value then
8:              case.id ←
                e1.attributes[ i ].value + e2.attributes[ i' ].value +
                e1.attributes[ j ].value + e2.attributes[ j' ].value +
                e1.attributes[ k ].value + e2.attributes[ k' ].value
9:              case.events ← case.events ∪ {e1, e2}
10:             process.cases ← process.cases ∪ {instance}
11:         end if
12:     end for
13:     Log.processes ← Log.processes ∪ process
14: end for
```

Business process can subsequently be discovered from the MXML event logs by applying different well-known techniques and algorithms developed from the business process mining field [4].

4 Empirical study

The technique used to obtain event logs by discovering correlation sets has been empirically validated through an industrial case study by implementing and applying the technique to two traditional information systems. The empirical study has been rigorously planned and conducted by following the formal protocol for conducting case studies proposed by Runeson et al. [26]. The following sections present a detailed description of the main stages defined in the formal protocol: background, design, case selection, execution and data collection, analysis and interpretation, and finally validity evaluation.

All of the software artifacts involved in this case study and the tool developed to support the proposed approach are available online [9].

4.1 Background

The proposed approach focuses particularly on traditional (non-process-aware) information systems and focuses on the correlation of events in the appropriate process instance. As a consequence, the *object of the study* is the aforementioned technique and the *purpose of the study* is to demonstrate the feasibility of the technique in terms of its accuracy. The main research question of the study is therefore *MQ*.

MQ. Can the technique obtain correlation sets with which to generate event logs from a traditional system, which can then be used to discover the business processes supported by the system?

The study additionally evaluates two secondary research questions: *AQ1* and *AQ2*. *AQ1* evaluates the gain of this new approach with regard to the previous technique used to obtain event logs using a correlation event mechanism that only trusts in single isolated pieces of source code as correlation sets, which are defined by the user. This secondary question is evaluated by comparing the result of this study with the result obtained in a previous study that validated the previous approach using the same traditional information systems [6,7]. The *AQ2* question analyzes the time taken to discover correlations sets in order to discover whether the technique is scalable to huge data sets.

AQ1. How does this technique perform in comparison to the previously developed technique?
AQ2. How much time does the technique take to discover correlation sets as regards the size of the data sets?

4.2 Design

The design of the empirical study follows the *embedded and multiple case study* design according to the classification proposed by Yin [27]. On the one hand, the study is embedded since it considers several analysis units within each case. On the other hand, it is a multi-case study since it considers two different cases (i.e., two different traditional information systems). A multiple case study allows researchers to analyze within each setting and across settings (i.e., it analyzes several cases to understand the similarities and differences between them). On the contrary, a single case study only allows researchers to understand one unique, extreme or critical case.

In order to answer the main research questions, the study is designed following a qualitative research approach by comparing a reference model and that obtained. The study first considers the business process models previously provided by business experts (the reference models). The study then obtains an event log through the correlation set and compares the business process instances collected in the log together with the reference business process model. The comparison between models evaluates the degree of conformance of the model obtained as regards the reference model. This is done by scoring the number of relevant business activities in common with the reference business process model. Relevant activities are considered to conform to the reference process when they meet three conditions:

1. The first condition specifies that the activity must represent a real-life business operation within the reference business process model.
2. The second condition ensures that all the relevant activities preceding the evaluated activity must be recovered before the activity under evaluation. In order to fulfill this condition, the predecessor activities can be directly or indirectly connected to the activity under evaluation, i.e., there could be non-relevant activities intercalated between the evaluated activity and its predecessor relevant activities.
3. In a similar manner, the third condition ensures that all the subsequent activities must be directly (or indirectly) recovered from relevant activities.
4. Finally, the fourth condition specifies that all the data objects related to the activity under evaluation have been also recovered.

Some quantitative measures are additionally considered to aid the qualitative approach. These measures are: (1) the number of events in addition to (2) the number of candidate correlation attributes collected in the intermediate database; (3) the beta threshold defined in each case, which defines different analysis units; (4) the number of process instances in the final event log; (5) the time taken to discover the correlation set (see Algorithm 1); and finally (6) the aforementioned conformance ratio.

4.3 Case selection

The two traditional (non-process-aware) information systems under study are *AELG-Members* and *CHES*. The cases were selected by following a set of four case selection criteria (see Table 2). $C1$ is defined to ensure that the system selected is a real-life information system that is currently in the production stage and supports the business operation of an organization or company. $C2$ ensures that the system selected is a traditional information system with no built-in logging mechanism. $C3$ ensures that the system is sufficiently large to be able to draw representative conclusions, exceeding 20,000 lines of source code. $C4$ requires the system to be based on Java technology, since the supporting tool with which to instrument traditional systems was preliminarily developed for Java-based systems. Nevertheless, the technique is not specifically for Java-based systems, but is based on the concept of callable units, which can also be applied to other programming languages by adding a parser for those languages.

AELG-Members and *CHES* were selected after evaluating several available systems according to these criteria. On the one hand, *AELG-Members* is an author management system that supports the administration of an organization of Spanish writers. From a technological point of view, *AELG-Members* is a *Java* standalone application whose architecture follows the traditional structure in three layers [21]: (1) the *domain* layer supporting all the business entities and controllers; (2) the *presentation* layer dealing with the user interfaces; and (3) the *persistency* layer handling data access. The total size of the legacy system is 23.3 KLOC (thousands of lines of source code).

Table 2 Criteria for case selection

Id	Criterion for case selection
C1	It must be an enterprise system
C2	It must be a non process-aware information system
C3	It must be of a size not less than 20 KLOC
C4	It must be a Java-based system

On the other hand, *CHES* is a healthcare information system used in several Austrian hospitals in different areas of medicine (e.g., oncology, geriatrics, psychiatry, psychosomatic medicine) for the collection, storage, and graphical processing of medical and psychosocial data. *CHES* also provides graphical real-time feedback of patient-related data in order to individualize treatment strategies, and permits individual patient and treatment data (e.g., laboratory values, data from medical interventions, questionnaire data) to be recorded. The total size of *CHES* is 91.3 KLOC and is also a *Java* application following a *three-layer* architecture.

Appendix 1 shows the business process supported by the systems under study which are considered as the reference business process models. Figure 9 presents the reference model of the *AELG-Members* system containing the main business activities carried out by the writers' organization, which include among other things, author registration, importing author information from different sources, cancelation of memberships, author information management and payment of fees. The reference model of the *CHES* system is shown in Fig. 10 in Appendix 1. The main business activities of this process are grouped in three lanes. (1) The patient admission lane manages all the activities needed to provide the hospital with new patients. (2) The data collection lane is performed by (a) the hospital staff who send questionnaires to collect information about the state of patients, and (b) the patients who fill out the questionnaires. Finally (3) the data analysis lane contains business activities with which to analyze information from the questionnaires, and these activities allow the hospital staff to order treatments or interventions.

4.4 Execution and data collection

The case study was executed by semi-automating all the stages of the proposed technique with different supporting tools. The steps carried out during the execution were the following. These steps also collect certain data to be analyzed.

Table 3 Source code instrumentation according to the guidelines

	AELG-Members			CHES		
	Number of classes	Discarded classes	Instrumented classes (%)	Number of classes	Discarded classes	Instrumented classes (%)
Whole system	165	151	8.48	2,491	2,102	15.62
Layers (G1)						
Presentation	19	19	0.00	1,797	1,791	0.33
Domain	91	77	15.38	422	39	90.76
Persistency	55	55	0.00	272	272	0.00
Design patterns						
Observer (G2)	0	0	0.00	15	15	0.00
DAO (G3)	25	25	0.00	24	19	20.83
Abstract factory (G4)	12	4	66.67	0	0	0.00
Sigleton (G5)	13	12	7.69	17	14	17.65
Facade (G6)	26	26	0.00	51	50	1.96

4.4.1 Step 1: instrumentation of systems under study

Both systems under study were instrumented using the *Event Traces Injector* tool [6], which semi-automates the instrumentation of source code by considering relevant information provided by experts. This tool was modified to support the collection of additional relevant information from experts such as the candidate correlation attributes. Two experts (a systems analyst and a business expert) were selected from each of the organizations from which the systems under study were taken.

These experts used the guidelines (cf. Sect. 3.1.2) to determine the domain business source code to be instrumented. Table 3 shows the number of classes of source code that were instrumented in both systems. Table 3 shows the instrumentation information for three different viewpoints: the whole system, the three layers according to guideline G1, and according to the design patterns considered in the remaining guidelines. Table 3 provides, in columns: (1) the total number of classes for each of the aforementioned viewpoints, (2) the number of classes discarded from the instrumentation by business experts; and (3) the percentage of source code instrumented.

The instrumentation results show that most parts of the source code were discarded according the guidelines. In fact, the final percentage of instrumented classes was around 8 and 16 % for both systems. The persistency and presentation layer were completely discarded for both systems except for six classes in the *CHES* presentation layer. With regard to the design patterns, the experts selected business domain classes in line with the guidelines, with the exception of certain classes (see Table 3). For instance, in the case of *AELG-Members*, there was a class involved in a singleton pattern that was not discarded by experts. Furthermore, in the case of CHES, some classes involved in the singleton and facade patterns were not discarded. This was probably because the experts considered these classes to be relevant classes containing domain methods to be instrumented.

Table 4 Candidate correlation attributes selected in the study

Study ID	Attribute ID	Java class	Output method
AELG-Members	1	FeeVO	getIdAuthor
	2	AuthorVO	getId
	3	AuthorVO	isHistoric
	4	AuthorVO	getMemberNumber
	5	PublicAuthorVO	getId
	6	AuthorVO	getFees
CHES	1	Patient	getPatientId
	2	Patient	getSocialSecurity Number
	3	Patient	isActive
	4	Questionnaire	isCompletely Answered
	5	Questionnaire	toString
	6	HibernateObject	getId
	7	Intervention	getComment

With regard to the candidate correlation attributes to be collected together with events, six attributes were selected in the case of *AELG-Members* (see Table 4). Some attributes regarding the identification of author were selected first because the business process focuses on this entity (attributes 1–5). Other attributes related to fees were also selected since the experts expect process instances to end when an author's annual fees are paid (attributes 1 and 6).

Table 5 Source code instrumentation results

Feature	AELG-Members	CHES
LOC	23,339	91,266
# Java files	165	822
# Processed Java files	16	181
# Instrumented callable units	33	186
Static analysis time	4″ (4,091 ms)	2′47″ (166,686 ms)

Table 6 Data sets of events and correlation attributes considered in the study

Study ID	Data set ID	# Events	# Correlation attributes
AELG-Members	Small	2,432	10,412
	Medium	7,608	33,278
	Large	15,305	74,136
CHES	Small	9,361	5,236
	Medium	26,464	13,825
	Large	50,618	25,277

In the case of *CHES*, seven attributes were selected as candidate correlation attributes (see Table 4).Some attributes were selected to identify the patient, such as attributes 1–3. Other attributes like 4 and 5 were selected by the experts to detect process instances from the different questionnaires sent to each patient. Finally, attributes 6 and 7 were selected to detect the end of each treatment or intervention ordered for a particular patient.

Table 5 provides relevant information derived from the source code instrumentation: (1) the number of lines of source code; (2) the number of Java source code files; (3) the number of Java files modified according to the domain files selected; (4) the number of Java methods instrumented; and finally (5) the total time needed to obtain the instrumented version of each system.

4.4.2 Step 2: collection of events

The instrumented versions of *AELG-Members* and *CHES* were executed, and the events and candidate correlation attributes were stored in a *SQL Server 2005* database until sufficient data sets with which to conduct the study had been collected.

The different configurations were tested by considering three different sizes of data set (small, medium and large) (see Fig. 4). In the case of *AELG-Members,* the selected data sets contained more than 2,000, 7,000 and 15,000 events. In the case of the *CHES* system, the data sets collected contained more than 9,000, 25,000 and 50,000 events.

4.4.3 Step 3: discovery of the correlation set

Algorithm 1 was then applied to the data sets to discover the correlation set. Unlike previous stages, this algorithm was implemented by means of a set of stored procedures using PL/SQL which executes a set of queries from data sets (Table 6).

Since the beta threshold (Eq. 6) can be chosen by business experts [8], the algorithm was applied with four different values: 0.25, 0.5, 0.75 and 1. Table 7 presents the correlation sets and the time (s) taken to discover these correlations sets for each beta threshold value. The correlation sets for *AELG-Members* are described in Table 8, and those for the *CHES* are described in Table 9.

4.4.4 Step 4: generation of event logs, business process discovery and conformance checking

After the correlation sets had been discovered, an event log was generated for each of them by applying Algorithm 2. This algorithm was also implemented through PL/SQL procedures in order to achieve a quick access and management of data sets of events and correlation events.

Six event logs were obtained in total: four logs for the four different correlations sets discovered from *AELG-Members* (see Table 8) and two event logs for the correlation sets obtained from *CHES* (see Table 9). These six event logs were eventually analyzed and compared with the reference models (see Appendix 1). This was done using the *ProM* tool [24] to discover the respective business process models for each event log. This study particularly used the genetic mining algorithm implemented in *ProM* since, according to [28], the accuracy of the genetic algorithm makes it the most suitable for this purpose.

Finally, the conformance of each business process model with the reference model was analyzed according to the aforementioned qualitative research approach (cf. Sect. 4.2). Table 10 provides the conformance ratio values for all the business process models discovered using the correlation sets discovered in the previous stage.

4.5 Analysis and interpretation

This section analyses all the data collected during the execution of this study and obtains the evidence chains needed to answer the research questions previously established. A brief interpretation of the correlation sets is first provided (cf. Sect. 4.5.1). The business processes discovered from the event logs obtained using the correlation sets are then compared with the business processes that were discovered from event logs obtained by applying a previous simple correlation technique (cf. Sect. 4.5.2). Finally, the performance analysis of this technique is presented in terms of the time taken to discover correlation sets (cf. Sect. 4.5.3).

Table 7 Correlation sets and time taken to discover them for each data set and system

Data set	Correlation set				Time (s)			
	$\beta = 0.25$	$\beta = 0.5$	$\beta = 0.75$	$\beta = 1$	$\beta = 0.25$	$\beta = 0.5$	$\beta = 0.75$	β=1
AELG-Members								
Small	A	C	C	C	12	15	16	15
Medium	A	C	C	C	41	56	55	55
Large	B	C	D	D	113	150	151	147
CHES								
Small	E	F	F	F	44	38	38	38
Medium	F	F	F	F	93	90	93	91
Large	F	F	F	F	399	409	400	398

Table 8 Four correlation sets obtained for the AELG-Members system

Atomic conditions					
A	o: 1 = 1	p: 2 = 2	q: 4 = 4	r: 6 = 6	
B	o: 1 = 1	p: 2 = 2	q: 4 = 4	r: 6 = 6	
C	o: 1 = 1	p: 2 = 2	q: 4 = 4	r: 6 = 6	s: 5 = 5
D	o: 1 = 1	p: 2 = 2	q: 4 = 4	r: 6 = 6	s: 5 = 5
Complex conditions					
A	o $\wedge$ q	p $\wedge$ q			
B	o $\wedge$ q	p $\wedge$ q	r $\wedge$ q	r $\wedge$ p	
C	o $\wedge$ q	p $\wedge$ q	r $\wedge$ q	r $\wedge$ p	
D	o $\wedge$ q	p $\wedge$ q	r $\wedge$ q	r $\wedge$ p	o $\wedge$ s

Numbers 1–6 refer to attribute id of Table 4. Letters o–s refer to atomic conditions

Table 9 Two correlation sets obtained for the CHES system

Atomic conditions						
E	o: 1 = 1	π: 2 = 2	ρ:3 = 3	ς: 4 = 4	σ: 5 = 5	τ: 6 = 6
F	o: 1 = 1	π: 2 = 2	ρ: 3 = 3	ς: 4 = 4	σ: 5 = 5	τ: 6 = 6
Complex conditions						
E	o $\wedge\rho$	$\pi \wedge \rho$	$\pi \wedge \tau$			
F	o $\wedge\rho$	$\pi \wedge \rho$	$\pi \wedge \tau$	$\varsigma \wedge \sigma$		

Numbers 1–7 refer to attribute ID in Table 4. Greek letters o to τ refer to atomic conditions

Table 10 Conformance checking for business process models discovered using correlation sets

	Correlation sets for AELG-Members				Correlation sets for CHES	
	A	B	C	D	E	F
# Tasks in reference model	14	14	14	14	18	18
# Tasks in discovered model	13	12	12	9	17	15
Conformance ratio (%)	93	86	86	64	94	83

4.5.1 Interpretation of the obtained correlation sets

Table 7 summarizes the results obtained after carrying out the case studies. These results show the correlation sets (A, B, C and D for *AELG-Members*; and E and F for *CHES*), which were obtained for different combinations of data sets (small, medium and large) and beta values (0.25, 0.5, 0.75 and 1). Table 7 also shows the time taken to discover each correlation set, in addition to the particular atomic and conjunctive conditions of each set.

AELG-Members

After obtaining the respective event log and discovering the respective business process for the *AELG-Members* system, it was perceived that the most accurate correlation set was 'A', leading to the business process with the highest conformance degree (93 %), i.e., the business process discovered using set 'A' had the highest number of business activities in common with the reference business process model.

The same conclusion can be stated by analyzing the conditions of correlation set 'A' (see Table 7). Set 'A' is less restrictive (in comparison to the other sets) and contains fewer correlation conditions. Despite this, it contains all the atomic conditions needed to evaluate the identity of each writer (i.e., *getIdAuthor, gedId* and *getMemberNumber*). Set 'A' also contains the atomic condition needed to know when a fee is paid (i.e., *getFees*), which signifies that a particular process instance ends for a writer.

Moreover, with regard to the complex conditions of correlation set 'A', there is a conjunctive condition that links *FeeVo.getIdAuthor* with *AuthorVO.getId*, which signifies that the fees managed must correspond to the same writer of a particular process instance. Finally, set 'A' also works well because the categorical correlation attribute *AuthorVO.isHistoric* was properly discarded, since these kinds of attributes (e.g., boolean variables) split the data sets into only two instances.

The remaining correlation sets (B, C and D) are similar to correlation set 'A', since all these sets contain all the correlation conditions of 'A'. However, these sets incorporate more conditions, and although they provide alternative event correlations, they are more restrictive. This means that some process instances obtained using 'A' could be split into two or more instances if sets B, C or D were used as the correlation set instead of set 'A'. These sets led to conformance values of between 64 and 86 % (see Table 10).

The correlation set 'A' was only obtained by employing a restrictive value for beta (i.e., 0.25) (see Table 7). In spite of all this, the lower beta value combined with the usage of the largest data set generated the correlation set 'B', which adds some conjunctive conditions with regard to A. The least suitable correlation sets, C and D (since these sets contain more conditions), were consequently obtained for higher values of the beta threshold.

CHES

After comparing the two business processes discovered in the two respective event logs (generated using correlation sets E and F, respectively), it was noted that the most accurate correlation set was set 'E' (see Tables 7, 9). Set 'E' led to a conformance value of 94 %, while the conformance of the business process model discovered using set 'F' was 83 % (see Table 10). Although the atomic conditions of both sets are the same, set 'E' is less restrictive than set 'F' since it contains only three conjunctive conditions. The sole difference between both sets is a conjunctive condition in set 'F': $(Questionnaire.toString = Questionnarie.toString) \wedge (Questionnarie.isComplete = Questionnarie.isComplete)$. However, this conjunctive condition is not necessary because it is formed of an atomic condition that evaluates the equality of a categorical correlation attribute such as *Questionnarie.isComplete*. This attribute is used to indicate whether a questionnaire has (or has not) been completely filled out by a patient.

In a similar way to the results obtained for the *AELG-Members* system, although set 'E' was the most accurate correlation set discovered, it was only obtained for the small data set and the lowest beta value (0.25). Indeed, the most common result was the correlation set 'F', since this set was systematically obtained for most configurations of data sets and beta values (see Table 7). Although set 'F' is not the most accurate correlation, it does not lead to inaccurate event logs. In fact, the comparison of sets 'E' and 'F' shows that the sole difference was a conjunctive condition.

Another important point is that the correlation set variability in the case of the *CHES* is lower than that of the *AELG-Members*. This may be because the candidate correlation attributes, in the case of the *CHES*, were probably better selected by the experts. This selection by experts can be considered as a threat to the validity (cf. Sect. 4.6).

4.5.2 Comparison with results of the previous technique

A previous technique used to obtain event logs from traditional (non-process-aware) information systems developed a preliminary and coarse event correlation mechanism. This mechanism considers a particular source code object to correlate all the events. The business processes discovered from the event logs obtained from the same cases in previous studies are provided in [6,7].

AELG-Members

In the previous case study with the same system [7], the Java class *'AuthorVO'* was selected as the classifier with which to collect correlation information during the system instrumentation stage. During system execution, the runtime values of the *AuthorVO* objects were used to correlate events. As a result, all the process instances in the event log were obtained with all the events regarding each writer. Unlike the current approach, not all the different executions of the reference business process (see Fig. 9) for each author were detected. For example, every time a writer pays his/her annual fee, it should be detected as the end of a process instance. This kind of aggregation works using any correlation set obtained with the current approach. However, the previous approach did not permit all the events of the same writer to be grouped into fine-grained process instances, since the sole information used to correlate events was the *AuthorVO* objects.

As a result, the business process model discovered in the event log of the previous study was less accurate (i.e., obtained with lower conformance degrees) than the current one, i.e., the preliminary model was obtained with various erroneous activities and other missing activities with regard

to the reference business process model. In total, the conformance degree was 77 % [7], while that obtained with the proposed technique is 93 % (obtained with set 'A', see Table 10). The business process obtained with the previous technique was therefore quite complex, and was visually considered to be a *spaghetti* model owing to several crossing sequence flows.

CHES

In the case of the *CHES*, the previous study [6] considered the Java class *'Patient'* as the classifier to be collected as the correlation information during system execution. Every process instance in the event log was therefore obtained with all the events regarding each patient. With the previous technique, all the different executions of the reference business process (see Fig. 10) for each patient were not detected, but all the events for the same patient were grouped together. For example, when a new treatment or intervention is ordered for a particular patient, all the events generated should be grouped in a different process instance. This correlation strategy works using any correlation set (sets 'E' or 'F') obtained with the current approach but not with the previous approach.

Similarly to the results obtained for the *AELG-Members*, all the events of the same patient cannot be grouped into fine-grained process instances because the sole correlation data set is the occurrence of *Patient* objects. The business process model discovered form the event log of the previous study was therefore less accurate than the current one. In fact, the preliminary model was also obtained with various erroneous and missing activities as regards the reference model. The sequence flow of the business process model obtained for the *CHES* was also quite intricate. In total, the conformance degree obtained with the previous technique was 64 % [6], while the conformance degree obtained using the new technique is 94 % (see Table 10).

4.5.3 Performance analysis

In order to demonstrate the feasibility of the proposal, the time taken to discover correlation sets was analyzed according to the additional question *AQ2* (cf. Sect. 4.1).

Moreover, outlier beta values (i.e., 1, and especially 0.25) lead to shorter times (see Table 7). This is owing to the fact that outlier values allow the algorithm to quickly prune non-promising correlation sets, which saves a considerable amount of time. It should be noted that the time regarding the beta value is approximately linear.

On the other hand, with regard to the number of events, the time is non-linear. Figure 8 shows the box charts of the time taken to discover correlation sets for each data set (small, medium and large) in both studies. The time is lower for smaller data sets and higher for larger ones. It would appear that the trend of the time follows a quadratic function. This

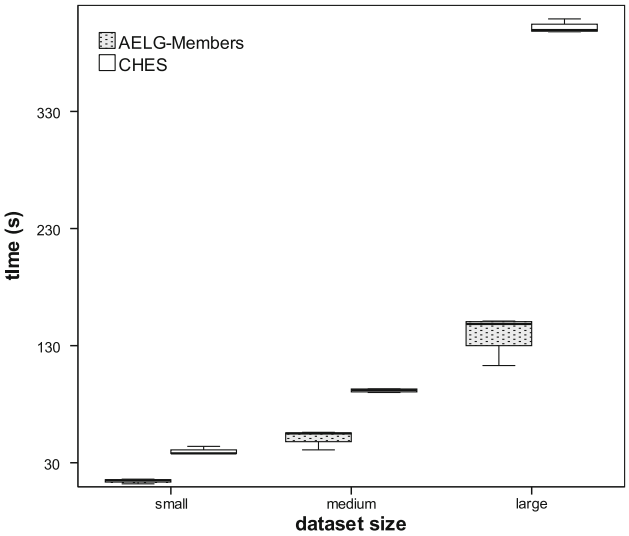


Fig. 8 Box chart for time taken to discover correlation sets in both cases

is owing to the fact that every event must be checked for all the remaining events according to the proposed algorithm.

In conclusion, the main research question can be positively answered. This signifies that the technique is able to correlate events from traditional systems, and it in turn produces a gain as regards the techniques previously developed. However, the time taken to discover the correlation sets is quadratic, and huge data sets may be time-consuming.

4.6 Validity evaluation

Finally, the validity of the results had to be assessed as unbiased and true for the whole population to which we wished to generalize the results. This section presents the threats to the validity of this case study and possible actions to mitigate them. There are three principal types of validity: internal, construct and external.

4.6.1 Internal validity

The most important threat to the internal validity is the fact that the code could be poorly instrumented. The results obtained clearly depend on the candidate correlation attributes selected at the beginning of the study. If business experts select an incomplete or erroneous set of candidate correlation attributes, the outgoing results could be quite different. In order to mitigate this treat we propose repeating the study using an iterative approach, in which experts can iteratively select or remove certain candidate correlation attributes according to the results obtained for each iteration. The list of candidate correlation attributes can thus be iteratively refined. Alternatively, some additional guidelines will be considered by considering other kinds of architectures and platform of legacy information systems such as aspect-oriented systems,

non-object-oriented systems, or even different programming languages to be instrumented.

4.6.2 Construct validity

Correlation sets do not always have to be obtained under lower beta values (e.g., 0.25). A lower beta value often implies a more restrictive correlation set and vice versa. The beta threshold can therefore be established by business experts depending on the constrain degree to be applied to the particular set of candidate correlation attributes. This threat can be addressed by repeating the study with different cases and different beta values.

Another threat to the construct validity is the fact that the execution of the instrumented systems was not carried out in a production environment, and the environment was instead simulated to collect events. This threat could be dealt with by executing the instrumented systems in their real environments.

4.6.3 External validity

Since the study was conducted with only two cases, the results obtained cannot be generalized to the entire population, i.e., to all non-process-aware information systems. This threat should be mitigated by carrying out more case studies involving different systems.

5 Conclusions

This paper addresses the problem of obtaining event logs from traditional (non-process-aware) information systems, which do not have any in-built mechanisms with which to record events. Furthermore, it particularly deals with the event correlation challenge. This challenge attempts to allocate events that have occurred to one of the process instances that are being executed in a particular moment by a traditional system.

This paper contributes to the solution of the aforementioned problem by presenting a technique with which to obtain event logs from traditional systems, which correlates events. The technique mainly consists of four stages. The first stage analyzes and instruments the source code of the traditional system so that the event collection can be automated. This stage provides some guidelines to aid experts' decisions as to which parts of source code have to be instrumented. The second stage collects events by means of the execution of the instrumented system. The third stage discovers the correlation set of attributes and conditions. Finally, the fourth stage generates event logs by allocating each event according to the correlation set discovered.

The main benefit of this approach (in comparison with inspecting the software directly in order to understand how it supports particular business processes) is that it provides a dynamic inspection of the business process traces. Dynamic analysis allows non-executed paths, dead code, etc. to be discarded. The new approach provides a better event correlation by choosing between certain candidate choices, which involves obtaining more accurate and consistent event logs, and thus also mining more accurate business processes.

The most important contribution of this paper is an empirical study conducted to demonstrate the feasibility of the technique in addition to its application in industry. The study was applied to two traditional industrial information systems: *AELG-Members*, an author management system and *CHES*, a system for collecting patient reported outcome (PRO) data. The study collected thousands of events for later analysis, and obtained different correlation sets according to different parameters. The correlation sets were then used to generate different event logs, which were in turn used to discover business processes. Finally, the mined business process models were compared with the reference models.

The analysis of the study's results shows that the technique is able to obtain event logs from traditional systems. However, the correlation set depends on the amount of events collected and the beta factor (which determines the way in which process instances are built). Although more collected events may be a good means to obtain better correlation sets, the time taken to discover this information increases according to a quadratic function. Our work-in-progress is therefore currently focused on obtaining correlation sets in a more efficient manner (to reduce the response time of the discovery algorithm) and on effectiveness (to provide accurate correlation sets with fewer events).

The main implications of this work are that event logs, which are the input for most business process mining techniques, can be obtained from traditional information systems. All the effort of the business process mining field can therefore be reused in traditional information systems, which are common in most companies and organizations. This work thus contributes towards improving enterprise modeling efforts in those companies that wish to start modeling their enterprise environments for the first time, since they can use business process mining from their existing information system.

Acknowledgments This work was supported by the FPU Spanish Program and the R&D projects ALTAMIRA (PII2I09-0106-2463), PEGASO/MAGO (TIN2009-13718-C02-01), MAESTRO (Alarcos Quality Center) and MOTERO (JCCM and FEDER, PEII11-0366-9449). Additionally, this work was supported by the University of Innsbruck.

Appendix 1: Reference and discovered business process models

This appendix provides the reference business process models that are supported by the two traditional information systems under study. Owing to space limitations, each figure depicts the reference model and different variations of the models discovered as regards the reference model in terms of business tasks that were not discovered by means of the different correlations sets.

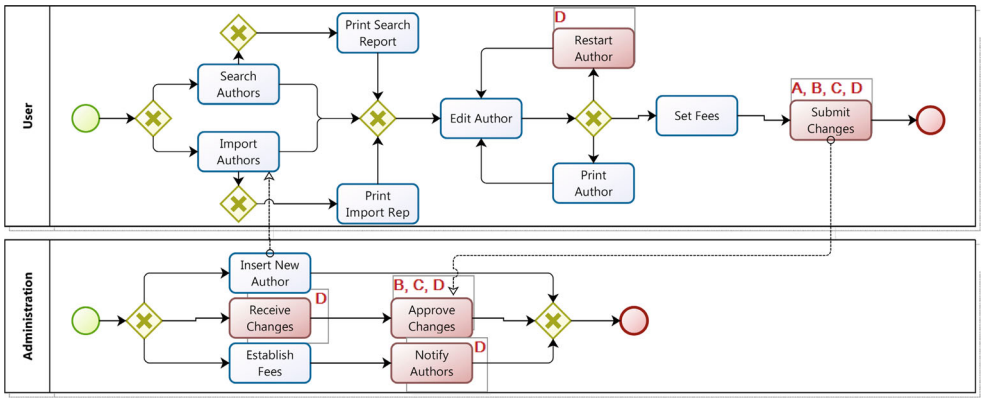


Fig. 9 Reference and discovered business process models of the *AELG-Members* system

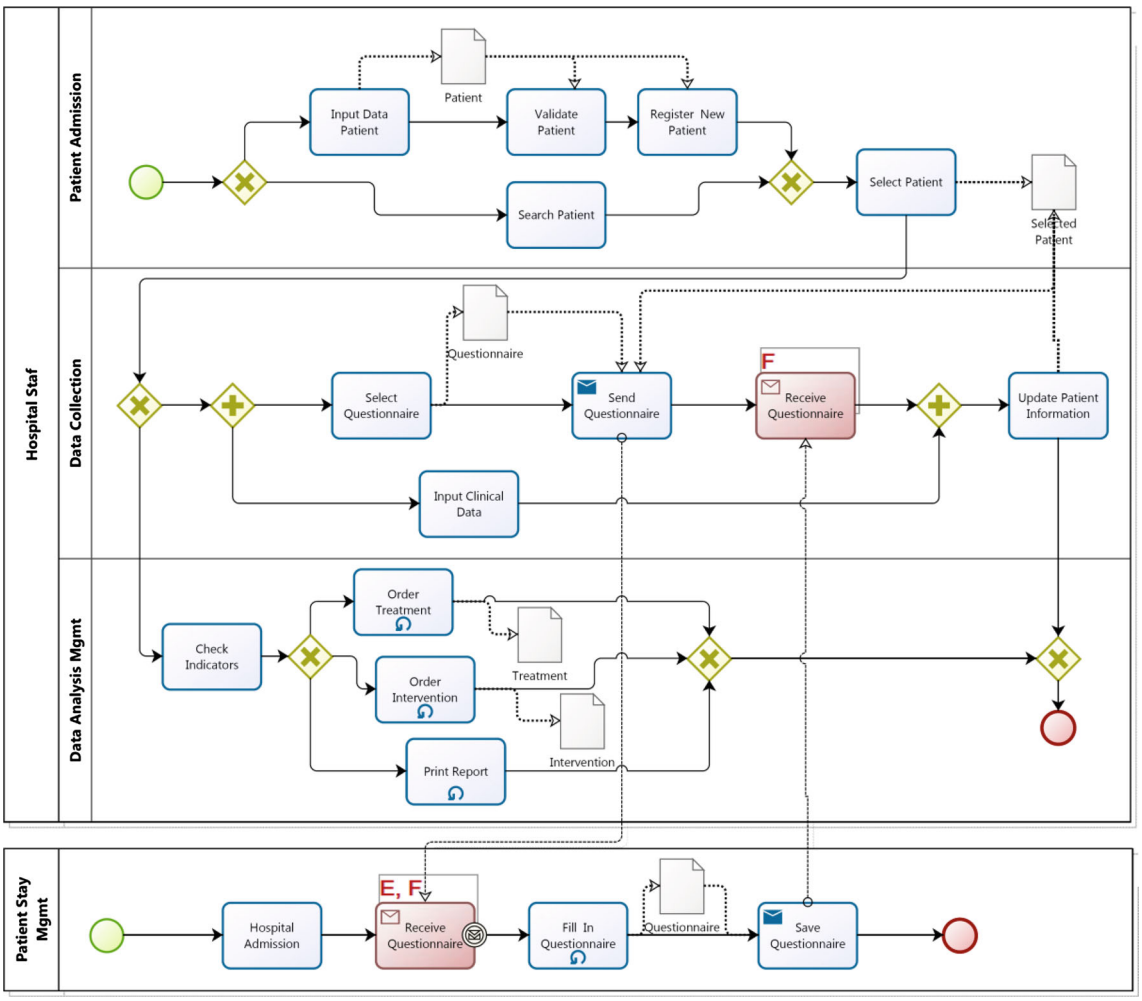


Fig. 10 Reference and discovered business process models of the *CHES*

Firstly, Fig. 9 depicts the reference model of the AELG-member systems and presents the four variations of business process models discovered using the correlation sets A, B, C and D (cf. Sect. 4.4). The missing business tasks for each correlation data set are highlighted and the correlation set ID are specified (A, B, C and D).

Secondly, Fig. 10 provides the same information for the CHES system. In this case, two different business process models were discovered using correlation sets E and F (cf. Sect. 4.4).

References

1. Buckl, S., et al.: A meta-language for enterprise architecture analysis. In: Halpin, T. (ed.) Enterprise, Business-Process and Information Systems Modeling, pp. 511–525. Springer, Berlin (2011)
2. Barn, B., Clark, T.: Revisiting Naur's programming as theory building for enterprise architecture modelling. In: Mouratidis, H., Rolland, C. (eds.) Advanced Information Systems Engineering, pp. 229–236. Springer, Berlin (2011)
3. Paradauskas, B., Laurikaitis, A.: Business knowledge extraction from legacy informations systems. Inf. Technol. Control **35**(3), 214–221 (2006)
4. van der Aalst, W., Weijters, A.J.M.M.: Process mining. In: Dumas, M., van der Aalst, W., Ter Hofstede, A. (eds.) Process-aware Information Systems: Bridging People and Software Through Process Technology, pp. 235–255. Wiley, New York (2005)
5. Pérez-Castillo, R., et al.: Toward Obtaining Event Logs from Legacy Code. Business Process Management Workshops (BPI'10). Lecture Notes in Business Information Processing (LNBIP 66–Part 2), pp. 201–207 (2010)
6. Pérez-Castillo, R., et al.: Generating event logs from non-process-aware systems enabling business process mining. Enterp. Inf. Syst. J. **5**(3), 301–335 (2011)
7. Pérez-Castillo, R., et al.: Process mining through dynamic analysis for modernizing legacy systems. IET Softw. J. **5**(3), 304–319 (2011)
8. Motahari-Nezhad, H.R., et al.: Event correlation for process discovery from web service interaction logs. VLDB J. **20**(3), 417–444 (2011)
9. Pérez-Castillo, R.: Experiment results about assessing event correlation in non-process-aware information systems (2012). http://alarcos.esi.uclm.es/per/rpdelcastillo/CorrelationExp.html#correlation
10. Ingvaldsen, J.E., Gulla, J.A.: Preprocessing support for large scale process mining of SAP transactions. Business Process Intelligence Workshop (BPI'07). In: LNCS, vol. 4928, pp. 30–41 (2008)
11. Günther, C.W., van der Aalst, W.M.P.: A generic import framework for process event logs. Business Process Intelligence Workshop (BPI'06). In: LNCS, vol. 4103, pp. 81–92 (2007)
12. McGarry, K.: A survey of interestingness measures for knowledge discovery. Knowl. Eng. Rev. **20**(1), 39–61 (2005)
13. Burattin, A., Vigo, R.: A Framework for Semi-Automated Process Instance Discovery from Decorative Attributes. In: IEEE Symposium on Computational Intelligence and Data Mining (CIDM'11), pp. 176–183. Paris, France (2011)
14. Rozsnyai, S., Slominski, A., Lakshmanan, G.T.: Discovering Event Correlation Rules for Semi-Structured Business Processes. In: Proceedings of the 5th ACM international conference on Distributed event-based system, pp. 75–86. ACM, New York (2011)
15. Ferreira, D., Gillblad, D.: Discovering process models from unlabelled event logs. In: Dayal, U. (ed.) Business Process Management, pp. 143–158. Springer, Berlin (2009)
16. Kato, K., Kanai, T., Uehara, S.: Source code partitioning using process mining. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) Business Process Management, pp. 38–49. Springer, Berlin (2011)
17. Myers, J., Grimaila, M.R., Mills, R.F.: Adding Value to Log Event Correlation Using Distributed Techniques. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, pp. 1–4. ACM, Oak Ridge (2010)
18. Hammoud, N.: Decentralized Log Event Correlation Architecture. In: Proceedings of the International Conference on Management of Emergent Digital EcoSystems, pp. 480–482. ACM, France (2009)
19. Zou, Y., Hung, M.: An Approach for Extracting Workflows from E-Commerce Applications. In: Proceedings of the Fourteenth International Conference on Program Comprehension. IEEE Computer Society, pp. 127–136 (2006)
20. Ratiu, D.: Reverse Engineering Domain Models from Source Code. In: International Workshop on Reverse Engineering Models from Software Artifacts (REM'09), pp. 13–16. Simula Research Laboratory, Lille, France (2009)
21. Eckerson, W.: Three tier client/server architecture: achieving scalability, performance and efficiency in client server applications. Open Inf. Syst. **10**(1), 3 (1995)
22. Gamma, E., et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Longman Publishing Co. ed., Inc., Boston, Addison Wesley, USA (1995)
23. Oracle Inc. Core J2EE Patterns: Data Access Object (http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html). Core J2EE Pattern Catalog 2001 [cited 11/04/2012]
24. Van der Aalst, W.M.P., et al.: ProM : The Process Mining Toolkit. In: 7th International Conference on Business Process Management (BPM'09)–Demonstration Track, pp. 1–4. Springer, Germany (2009)
25. Fluxicon Process Laboratories, XES 1.0 Standard Definitio (Extensible Event Stream). http://www.xes-standard.org/ (2009)
26. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Eng. **14**(2), 131–164 (2009)
27. Yin, R.K.: Case Study Research. Design and Methods, 3rd edn. Sage, London (2003)
28. Medeiros, A.K., Weijters, A.J., Aalst, W.M.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Discov. **14**(2), 245–304 (2007)

Author Biographies

Ricardo Pérez-Castillo holds the Ph.D. degree in Computer Science from the University of Castilla-La Mancha (Spain). He works at the Information Systems and Technologies Institute at University of Castilla-La Mancha. His research intererests include architecture-driven modernization, model-driven development, business process archeology and service science. Ricardo has published more than 40 refereed papers, for example, in Enterprise Information Systems, Systems and Software, Information & Software Technology, Computer Standards & Interfaces, among others.

Barbara Weber is an associate professor at the University of Innsbruck (Austria). Barbara is a member of the Quality Engineering (QE) Research Group and head of the Research Cluster on Business Processes and Workflows at QE. Barbara holds a Habilitation degree in Computer Science and Ph.D. in Economics from the University of Innsbruck. Her main research interests are agile and flexible processes, integrated process lifecycle support, intelligent user support in flexible systems and process modeling. Barbara has published more than 70 refereed papers, for example, in Data & Knowledge Engineering, Computers in Industry, Science of Computer Programming, Enterprise Information Systems and IET Software. Moreover, Barbara is organizer of the successful BPI workshop series.

Ignacio García-Rodríguez de Guzmán is assistant professor at the University of Castilla-La Mancha (Spain) and belongs to the Alarcos Research Group at the UCLM. He holds the PhD degree in Computer Science from the University of Castilla-La Mancha. His research interests include software maintenance, software modernization and service-oriented architecture.

Mario Piattini is full professor at the University of Castilla-La Mancha (Spain). His research interests include software quality, metrics and maintenance. He holds the Ph.D. degree in Computer Science from the Technical University of Madrid, and leads the Alarcos Research Group at University of Castilla-La Mancha. He is certified as CISA, CISM e CGEIT by ISACA.

Jakob Pinggera is a Ph.D. candidate at the University of Innsbruck (Austria). Jakob is a member of Quality Engineering (QE) Research Group and member of the Research Cluster on Business Processes and Workflows at QE. Jakob received his M.Sc. degree from the Department of Computer Science, University of Innsbruck in 2009. His main research interest is the creation of business process models. Jakob has published more than 20 refereed papers in international journals, conferences and workshops.