

CONCEPT LOCATION MODELING THROUGH BUSINESS PROCESS VIEWS

RICARDO PÉREZ-CASTILLO* and MARIO PIATTINI†

*Alarcos Research Group, University of Castilla-La Mancha
Paseo de la Universidad, 4 13071, Ciudad Real, Spain*

**ricardo.pdelcastillo@uclm.es*

†mario.piattini@uclm.es

BARBARA WEBER

*University of Innsbruck
Technikerstraße 21a, 6020 Innsbruck, Austria
barbara.weber@uibk.ac.at*

Received 2 November 2011

Accepted 6 December 2012

Published 10 April 2013

Concept location is a key activity during software modernization since it allows maintainers to exactly determine what pieces of source code support a specific concept. Real-world business processes and information systems providing operational IT support for respective processes can be misaligned as a consequence of uncontrolled maintenance over time. When concepts supported by an information system are getting outdated or misaligned, concept location becomes a time-consuming and error-prone task. Moreover, enterprise information systems (which implement business processes) embed significant business knowledge over time that is neither present nor documented anywhere else. To support the evolution of existing information systems, the embedded knowledge must first be retrieved and depicted in up-to-date business process models and then be mapped to the source code. This paper addresses this issue through a concept location approach that considers business activities as the key concept to be located and discovers different partial business process views for each piece of source code. Thus, the concept location problem becomes the problem of extracting such views. This approach follows model-driven development principles and an automatic model transformation is implemented to facilitate its adoption. Moreover, a case study involving two real-life information system demonstrates its feasibility.

Keywords: Business process views; concept location; maintenance; knowledge discovery metamodel; model driven development.

1. Introduction

As a consequence of uncontrolled maintenance, information systems, and software in general, age over time becoming legacy information systems (LISs). When the maintainability of LISs decreases under acceptable limits companies must replace or modernize their LISs according to their current business processes to keep their competitiveness level.¹ Unfortunately, the companies' business process models, which

depict a set of coordinated activities and tasks to achieve their business objectives,² are often outdated and are not aligned with the actual business processes supported by their LISs. This is due to the fact that LISs are maintained to support the progressive business process evolution motivated by the need of retaining the competitiveness level in changeable environments. As a result, LISs progressively embed much valuable business knowledge during the system maintenance.³ In addition, the embedded business knowledge is neither present nor documented anywhere else. When LISs are replaced or modernized, their embedded business knowledge has therefore to be preserved in the evolved information systems, and business processes models must be updated with the embedded knowledge representing the real-world business processes.⁴ Preservation of embedded business knowledge entails at least two main challenges. The first challenge is the *discovery* or elicitation of business knowledge itself and the second challenge consists of the *effective usage* of the discovered knowledge to take advantage during evolution and modernization of LISs.

The first challenge (i.e. discovery of the embedded business knowledge) can be addressed through some techniques related to business process mining.⁵ It seeks to extract business process knowledge from process execution logs known as event logs, which contain information about the start and completion of activities executed by the processes.⁶ Most process mining techniques and algorithms take, as input, event logs obtained from process-aware information systems (PAIS),⁷ e.g. process management systems such as enterprise resource planning (ERP) or customer relationship management (CRM) systems, whose nature (in particular their process-awareness) facilitates the direct registration of events throughout process execution. Indeed, event logs are represented in a common format used by the majority of process-mining tools known as MXML (Mining XML).⁸ However, LISs (which mainly are the focus of our work) typically are not process-aware and respective process mining techniques cannot be directly applied. In previous work, we consequently developed a technique for obtaining event logs from LISs using dynamic analysis^{9,10} making business process mining techniques applicable to LISs that necessarily are not process-aware systems.

The second mentioned challenge concerns the effective usage of the discovered business knowledge in order to achieve more effective LIS modernization processes. All functionalities or services implemented in the enhanced information systems must support, or be aligned with, the actual business processes previously discovered.⁴ As a consequence, the key activity in this scenario is *concept location*¹¹ (also known as feature location) in order to answer questions like *what pieces of source code of a LIS supports a particular part or concept of a business process?* Moreover, concept location can also be used to analyze *what concepts of the actual business process are supported by a particular piece of source code?*

This paper focuses on both challenges although its main contribution is a concept location approach regarding second challenge since business process discovery has been addressed in previous work.^{12,9,10,13} The approach consists of an automatic

concept location technique through the mapping between pieces of legacy source code, represented by a code model, and partial views of business process models discovered from event logs. In general terms, a process view allows an abstraction from undesired details by omitting or even aggregating activities.¹⁴ A process view results from specific transformations applied to a process model.¹⁵ In particular terms, business process views are partial representations of a business process model in terms of some properties,¹⁶ for example a view grouping the set of business activities carried out by a specific supporting tool. Business process views facilitate the location of business process model's concepts that are supported by particular pieces of source code.

The approach can be used in two different scenarios. First, when a real-world process evolves and maintainers therefore want to propagate this change to the LIS, our concept location approach can determine what concepts of the business process are supported by a particular piece of code. It reduces maintenance efforts and the maintenance costs therefore are also lower. Second, when maintainers do not have any concept location information yet but the LIS must be re-implemented as a result of a business process change, our approach facilitates both the discovery of up-to-date business process models as well as concept location information. This location information is especially important when system is being modernized, since maintainers know the certain piece of source code that should be replaced or improved so that it supports the change in a particular business process activity. Moreover, this information can then be used in future maintenance projects (i.e. Scenario 1).

The remaining of the paper is organized as follows. Section 2 summarizes work related to our proposal. Section 3 presents in detail our concept location approach based on the discovery of process views. Section 4 provides an industrial case study involving two LISs. Finally, Sec. 5 provides a conclusion and discusses future research lines.

2. Related Work

This section summarizes work related to business process mining techniques to discover business processes (see e.g. Sec. 2.1) as well as related to concept location techniques (see e.g. Sec. 2.2).

2.1. *Business process mining*

There exists much work concerning business process mining. Most proposals provide mining techniques or algorithms to obtain business processes from event logs. For example, Van der Aalst *et al.*¹⁷ proposed the α -algorithm to discover the control flow of business processes from event logs. Similarly, Madeiros *et al.*¹⁸ suggested a genetic algorithm for business process discovery. Other approaches focus on the registration of event logs, e.g. Ingvaldsen *et al.*¹⁹ focus on ERP systems to obtain

event logs from the SAP’s transaction data logs. Günther *et al.*²⁰ provide a generic import framework for obtaining event logs from different kinds of PAIS. Typically, process mining algorithms assume the presence of a PAIS which registers the events regarding the execution of a business process in events logs. Pérez-Castillo *et al.*,⁹ in turn, propose an approach to obtain event logs by means of the injection of traces in source code to enable the registration of event logs for traditional nonprocess-aware systems. This work facilitates the application of the aforementioned algorithms to any kind of information system.

There are additional approaches to discover business processes by directly applying reverse engineering techniques on LISs, i.e. without the consideration of event logs [see Fig. 1(a)]. For example, Zou *et al.*²¹ developed a framework that statically analyzes the legacy source code and applies a set of heuristic rules to recover the underlying workflows. Other work focuses on recovering business processes by dynamically tracing the system execution driven by use cases (e.g. Cai *et al.*²²) or driven by the users’ navigation in graphical user interfaces (e.g. di Francescomarino *et al.*²³).

Figures 1(a) and 1(b) provides a comparison of static and dynamic approaches for discovering business processes. While static analysis exhaustively analyzes programming language descriptions line by line, dynamic analysis focuses on actual information that can only be known at runtime. Static approaches are able to discover large business processes, since the structure of the entire LIS can be represented as code models. However, business processes obtained using static analysis are less accurate than processes obtained using dynamic approaches, since dynamic ones consider system execution information like event logs, which discards exceptional or dead parts of the source code. In fact, event-log-supported techniques are suitable to provide compact and accurate business processes. It is due to the fact

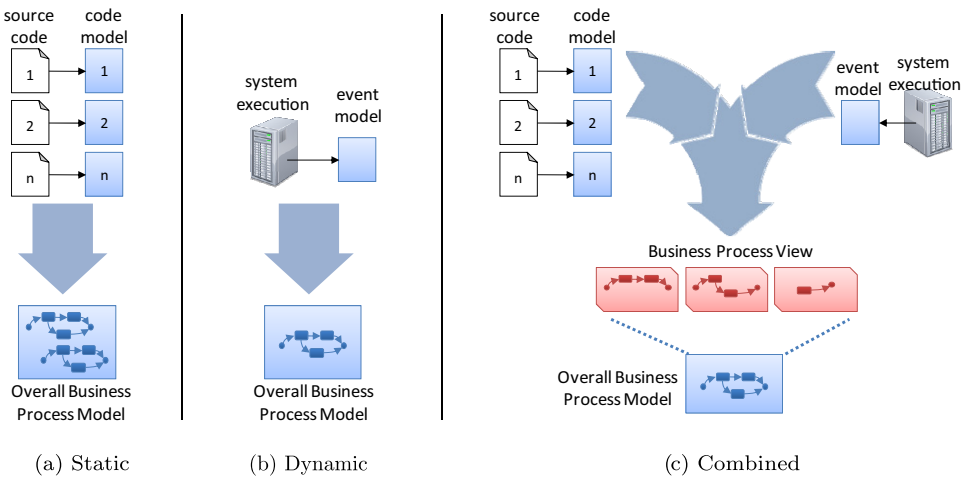


Fig. 1. Comparison of three approaches to discover business processes.

that dynamic analysis only considers the business activities that are actually carried out through the system execution.

2.2. Concept location and process views

Several approaches for concept location during the maintenance stage have been proposed in literature. Wilde *et al.*²⁴ proposed *software reconnaissance*, a dynamic approach to locate user functionalities into a LIS through the analysis of the execution of different test cases. This proposal provides various views of software that maps features to program components at different abstraction levels, although it ignores the business knowledge level. Similarly, Eisenbarth *et al.*¹¹ locate features in source code by gathering information from a set of scenarios invoking the features. Rather than assuming a one-to-one correspondence between features and scenarios as in previous works, it manages scenarios that invoke many features, which is also made in our proposal. Marcus *et al.*²⁵ provided the latent semantic indexing (LSI) technique to map concepts expressed in natural language by the programmer to the relevant parts of the source code. Chen *et al.*²⁶ proposed a concept location technique based on an abstract system dependency graph that is statically derived.

The concepts of all these approaches are expressed in different ways (e.g. as a functionality, in natural language, etc.). Business process models (as used in our approach) can help to establish concepts by using the business process modeling and notation (BPMN) standard,²⁷ which is understood by business experts and maintainers. Furthermore, the obtained concept location is better during software modernization, since business activities represent concepts at a higher abstraction level. Motahari *et al.*¹⁶ also considered business activities as concepts since it obtains business process views. Contrary to our approach, Motahari *et al.* located business process views for each entire subsystem of the LIS. Our approach, however, maps different business process views to particular pieces of source code by combining artifacts obtained by both process mining approaches: the static and dynamic one [see Fig. 1(c)]. Eshuis *et al.*¹⁴ proposed a method to obtain process views from structured process models instead of from LIS. First, a noncustomized process view is constructed from an internal structured process model by aggregating internal activities the provider wishes to hide. Second, a customized process view is constructed by aggregating and omitting activities from the noncustomized view that are not requested by the consumer. Schumm *et al.*¹⁵ introduced a metamodel for process views as well as process viewing patterns which specify elementary transformations to alter an existing process. This work provides a platform-independent mechanism to obtain process views in a similar way that our proposal. However, it does not consider process views as a mapping between legacy source code and business concepts. Finally, Zhao *et al.*²⁸ proposed *FlexView*, a rigorous view model to specify the dependency and correlation between structural components of process views with emphasis on the characteristics of WS-BPEL. The disadvantage of this framework is that is technologically independent.

3. Concept Location Through Business Process Views

The problem of concept location identification is equated with the problem of business process view extraction. The following subsections present in detail the proposed concept location technique.

3.1. Overview

The proposed concept location approach is based on the discovery of business processes from LISs and the generation of particular process views. A *business process view*¹⁶ provides an abstract representation of one or more relevant business activities (e.g. managing patient) regarding tasks (e.g. patient admission, patient intervention), its control flow, and some particular properties (e.g. users, execution time, etc.). Business process views play a similar role than data views play in relational databases.

The concept location technique is aimed at knowing, through a business process view, what pieces of legacy source code support a specific fragment of the entire business process supported by the LIS. In addition, each business activity in a business process view contains the information about the fine-grained piece of source code that supports this activity. As a result, the concept location is achieved in two directions: source code is mapped with business process views, and business activities within process views are mapped with source code units.

While existing approaches either focus on static analysis [see Fig. 1(a)] or dynamic analysis [see Fig. 1(b)], the proposed technique obtains business process view models by combining both approaches. The input of the technique is two kinds of models: code models obtained using the static approach and event logs obtained through the dynamic approach [see Fig. 1(c)]. First, the code model represents an exhaustive structure of all the different parts of the legacy source code (see e.g. Sec. 3.2). Second, the event model depicts an event log, i.e. a certain execution sequence of business activities supported by the LISs (see e.g. Sec. 3.3). Furthermore, the concept location technique consists of pattern matching between both kinds of input models (see e.g. Sec. 3.4). Finally, pattern matching is automated by means of model transformation that applies the set of proposed patterns (see e.g. Sec. 3.5).

The technique follows the model-driven development principles, thus it uses particular metamodels to represent the involved models. Metamodels define the possible set of elements and its relationships for each kind of model. Each model must then conform to the metamodel established to create that kind of models. Particularly, both input models (i.e. code and event models) are represented according to the knowledge discovery metamodel (KDM)¹¹ which is a standard especially developed to represent models during the reverse engineering stage in a bottom-up manner. In contrast to unified modeling language (UML), KDM facilitates the representation of knowledge discovered from different legacy software artifacts and different viewpoints of the system (e.g. source code, databases, events, user interfaces).

In addition, KDM has become the common interchange format to share models about software artifacts between tools for modernizing and evolving LISs. The main advantage of KDM regarding other specifications such as UML is that it can be used in combination with related specifications of the architecture-driven modernization (ADM) initiative proposed by the OMG,²⁹ e.g. abstract syntax tree metamodel (ASTM) or software metrics metamodel (SMM). As a consequence, the approach uses KDM to ensure its applicability in industrial software modernization projects.

3.2. Code model

The *codemodel* represents a language-independent representation for various constructs defined by common programming languages. The *code model* depicts implementation level program elements and their associations from a static view point. A sole code model can represent all the source code of a whole LIS or there can be several code models representing different pieces of a certain LIS. In fact, code models could be fine-grained models (e.g. a model for each compilation unit) or could be coarse-grained models (e.g. a model for each subsystem). Software engineers make the decision about what parts of LISs must be grouped under the same code model to obtain a respective business process view.

3.2.1. Technique for obtaining a code model

Code models can be automatically obtained by statically analyzing source code. Static analysis is a reverse engineering technique based on compiler techniques such as parsing and data flow algorithms, which provide an abstract interpretation of the source code structure. Our approach uses the automatic static analysis technique described previously in Ref. 12, which syntactically analyzes the existing source code and progressively obtains a code model with all the information retrieved. This kind of analyses is automated by means of parsers that are built according to the particular grammar or metamodels. The static analysis is exhaustive since it analyzes all the programming expressions line by line. However, actual, run-time values are unknown. Thus, this technique following the static approach is not able to obtain information about runtime behavior. Despite this drawback, the main advantage of this technique is that it efficiently retrieves full knowledge of the LIS's structure [see Fig. 1(a)].

3.2.2. Metamodel for the code model

The syntactic analyzer especially developed for the programming language of the LIS represents the retrieved information in a standardized way according to the aforementioned KDM metamodel. The proposed technique particularly considers the specification of the *code* and *action* packages defined by the KDM standard.³⁰ Figure 2 summarizes the KDM metamodel concerning *code* and *action*

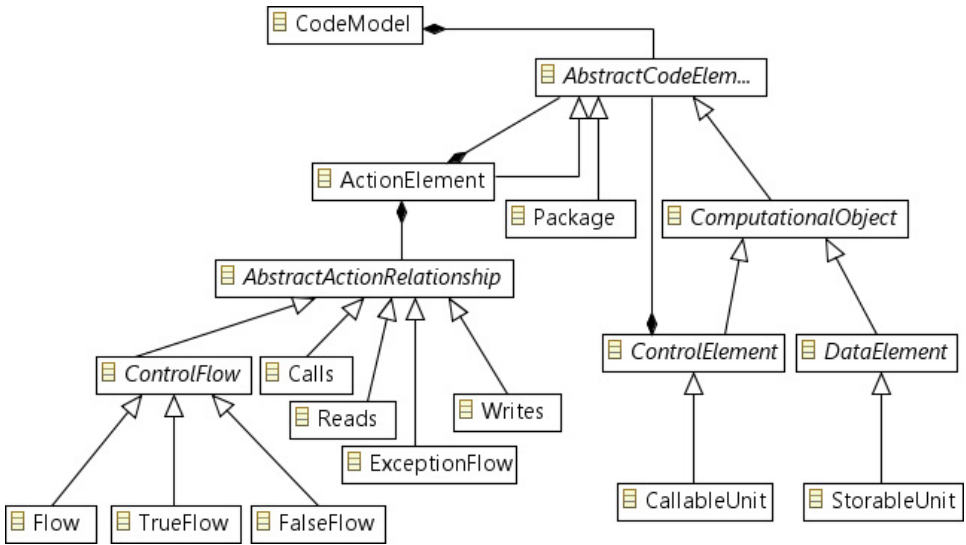


Fig. 2. The code metamodel according to the KDM standard.

packages in order to comprehend the information registered through a code model. A LIS is represented as a *CodeModel* element, the root metaclass. A *CodeModel* is in turn composed of *AbstractCodeElements*, a metaclass that represents an abstract class for all KDM entities that can be used as *CallableUnit*, *StorableUnit*, etc. *CodeElements* are also interrelated by means of *AbstractCodeRelationships*, a metaclass to represent an abstract class for all KDM relationships that can be used to represent the code such as *Flow*, *Calls*, *Reads*, *Writes*.

3.3. Event model

The *event model* represents an event log, which collects different sequences of business activities executed by the LIS (i.e. different business process instances). Each event model can contain thousands of business process instances. While the code model describes the LIS from a static point of view, the *event model* provides the dynamic viewpoint of the information system. In our combined approach, this model is aimed at discovering the business processes embedded in the LIS. On the contrary, code models allow maintainers filter out some elements of event models and obtain business process views according to certain pieces of source code. In contrast to the code models, the event model always represents information regarding the whole LIS. However, there could be various event models with information about system executions at different times. In addition, the execution of different information systems (which support the same business processes) can be registered in a sole event log.

3.3.1. Technique for obtaining event models

For PAIS the *event model* can be obtained automatically,⁵ since they have mechanisms to register event logs. However, for nonprocess-aware information systems reverse engineering techniques are needed to obtain event models. In previous work, we have developed a technique for automatically obtaining event models from traditional nonprocess aware information systems.¹⁰ This technique is based on dynamic analysis of the LIS and registers the execution of business process activities.

Since traditional information systems do not have any in-built mechanism to register events about executed business processes, this technique instruments such systems enabling them to register events. The instrumentation is semi-automated by a parser that syntactically analyzes de source code and injects statements in particular places of the code to register events during system execution. Statements are injected into callable units, i.e. Java methods, C procedures or Visual Basic functions (depending of the programming language). However, not all the executions of callable units are registered as events. Some callable units such as fine-grained or technical callable units (e.g. logging or data access modules) do not correspond to business activities and must be discarded. The injection in the appropriate place is consequently aided by some information provided by experts. Such experts (i) delimit business processes with the start and end callable units of each process; (ii) establish the boundaries of nontechnical source code to be instrumented; and finally, (iii) they identify those code elements that can be treated as correlation objects, which are used to build each process instance with their correct events. After that, the instrumented code is able to record event logs models during its execution.

3.3.2. Metamodel for event models

The *Event model* is represented according to the *event* metamodel package of the KDM specification³⁰ rather than the *code* and *action* one such as the code model. Figure 3 shows the *event* metamodel package of the KDM specification as well as other used meta-classes of other KDM packages needed to represent event models. The *EventModel* metaclass (the root metaclass) represents an event model which contains (i) one or more *EventResource* elements stereotyped as << *Process* >>, which represent the business processes supported by the LIS, and (ii) instances of the *EventAction* metaclass. *EventResource* metaclass is specialized into the *State*, *Transition* and *Event* metaclass, or even those *EventResource* elements stereotyped as << *Process* >> contain other *EventResource* elements stereotyped as << *Process Instance* >> which represent a particular execution sequence of business activities of the respective business process. The *Event* metaclass is finally used for modeling the event regarding the execution of a particular business activity supported by the LIS. The *Event* metaclass contains the features *name* representing the name of the executed business activity, and the feature *kind* to indicate if this event is refer to the start or end of the execution of this business activity. The

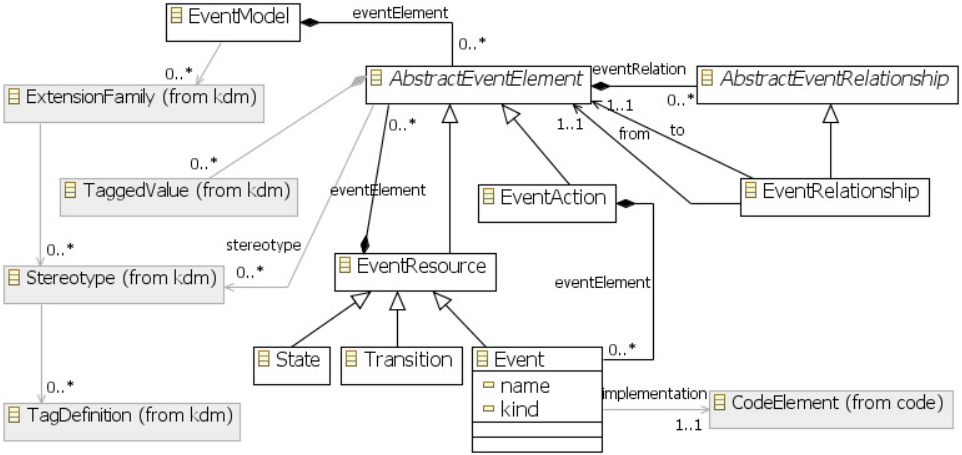


Fig. 3. The event metamodel according to the KDM standard and the extension mechanism.

Event metaclass is extended by means of tagged values to incorporate additional information regarding the *originator* and *timestamp*. Each instance of the *Event* metaclass is additionally linked to an instance of the *CodeElement* metaclass (from the code model) that represents the piece of source code that support the execution of the registered business activity.

3.4. Business process view generation

To obtain different business process models (representing process views) the approach provides a model transformation taking both input models into account: code and event models. The model transformation is considered as a *Y-transformation* (due to the underlying two-to-one relationship). The transformation takes two different input models and obtains a single output model [see Fig. 1(c)].

The model transformation supports pattern matching between these models. Pattern matching techniques check some sequence of tokens for the presence of the elements of some patterns. In contrast to pattern recognition, the match usually has to be exact. The patterns of this proposal are defined by means of either sequences or tree structures of elements of the code and event model.

The proposed model transformation is based on a preliminary set of business patterns previously provided in Ref. 12, which were developed to discover business processes from code models following a static approach. The main difference is that the previous transformation implements a one-to-one relationship between code model and business process model while the current one incorporates the event model as an additional input in a two-to-one relationship.

The previous patterns recognize certain structures in a source code model and infer specific elements in the target business process model (see Table 1 for an overview). For example, the *sequence* pattern recognizes callable units (e.g.

Table 1. Patterns to discover business processes from code and event model.


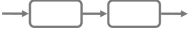

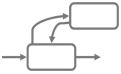

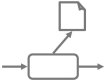





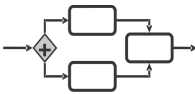
Pattern	Use	Description
P1. BPD Skeleton 	D	This pattern creates the root structure of the BP model. It creates a BP diagram for each KDM code model. Also, it builds a pool element with a nested process in the BP diagram for each package of the KDM code model.
P2. Sequence 	M	This pattern takes any callable piece of code from the KDM code model and maps them into tasks in the BP diagram. In addition, the sequence of calls to callable units is transformed into a set of sequence flows in the same order between the tasks built from the callable unit respectively.
P3. Branching 	D	This pattern transforms each conditional jump of the source code that has two mutually exclusive choices into an exclusive gateway and two different sequence flows in the BP model. Typically those exclusive conditional branches are related to the <i>if-then-else</i> or <i>switch</i> clauses in several programming languages. The exclusive gateway represents the condition that is evaluated and the two sequence flows represent two conditional transitions that depend on the value (true or false) of the evaluation.
P4. Collaboration 	M	Each call to external callable unit (i.e. API libraries or external components outside the legacy system) is transformed into an auxiliary task as well as two sequence flows: the first from the source task to the auxiliary task and the second returning to the source task.
P5. Data Input 	M	This pattern builds a data object in the BP model for each input data within a callable unit in the KDM code model. Also, this pattern builds an association between the data objects and the task previously built from the callable unit. This pattern only considers as input data the parameters or arguments of the callable unit, but it does not consider the auxiliary variables within the callable unit.
P6. Data Output 	M	Each piece of output data involved in a callable unit is transformed by means of this pattern into a data object as well as an association from the task (built from the callable unit) to the data object. This pattern excludes as output data the auxiliary and intermediate data in the body of the callable unit. The output data is the data returned by the callable unit or external data related to databases or files.
P7. Start 	D	The task building from the callable unit that starts the execution of any program or application of the legacy system is considered the initial task. Therefore, a start event is built into the BP diagram and a sequence flow from this event to the initial task is also created.
P8. Implicit Termination 	D	This pattern builds an end event in the BP model. Then, it creates sequence flows from “end task” and those flows merge in the end event. A task is considered an “end task” if this task does not have any outgoing sequence flow.

Table 1. (Continued)

Pattern	Use	Description
P9. Conditional Sequence 	M	This pattern transforms each conditional call into a sequence flow fired under a conditional intermediate event through to the task related to the callable unit. This pattern makes it possible to create arbitrary cycles in the BP diagram.
P10. Exception 	M	Each call to callable unit under any exception is transformed into a task for the piece of source code that handles the exception as well as a sequence flow fired under an error intermediate event. Indeed, this pattern can be understood as a specialization of the previous pattern P9.
P11. Process Instance 	A	Creates a sequence of business activities for each business process instances and interconnects them by means of sequence flows following the order of events in the business process instance.
P12. Process Instance Branching 	A	Create “and” gateways for each alternative sequence in different business process instances of the event model.

methods, functions, procedures, etc.) and the respective calls between them in the code model. That pattern then creates a new task in the business process model for each callable unit and a sequence flow for each call.

In order to combine static and dynamic approach and obtain business process views, the approach modifies some of the patterns provided in Ref. 12 (M), deprecates other ones (D) and incorporates additional patterns (A) taking information of the event model into account (see Table 1).

3.4.1. Deprecated patterns

Most deprecated patterns are discarded since the business process information that they consider can be discovered from event models with higher accuracy levels than from code models. For example, pattern P1 aimed at delimiting the scope of each business process (see Table 1) is deprecated because an event model can be used to obtain explicit definitions of each business process, thus, it is not necessary to infer business process structure. Patterns P7 and P8 are also discarded for the same reason since an event model contains the start and end points of business processes. This approach adds the new pattern P11 to recover the same required information from event models instead of code models which are statically obtained.

Pattern P3 (see Table 1) is also deprecated since the discovery of branches is based on choice or decision statements within code model. Unfortunately, it cannot know the result of the decision (i.e. the target statement where the execution flow will continue). As a consequence, the new pattern P12 is added considering the

event model's information to discover the choices and respective parallel branches of business processes. Consequently, in all these cases, it is more appropriate to directly take the information from the event model, instead of deducing that knowledge from code models by means of heuristics.

3.4.2. Modified patterns

Moreover, other patterns proposed in Ref. 12 have been modified (M) (e.g. Patterns P2, P4, P5, P6, P9 and P10 in Table 1). All these patterns are enhanced by considering a new important requisite: a business activity is created from a callable unit only when the respective business activity has been executed, i.e. it is registered in the event model. This means that code and event models are merged and only the business activities present in both models are added to the business process view. This is the mechanism to combine both kinds of models. Table 2 present a matrix between the input elements of code/event models as well as the output elements generated in business process models for each pattern modified or added.

3.4.3. Added patterns

Finally, Table 1 shows the new patterns added (A) to consider additional information of the event model (i.e. Patterns P11 and P12). Pattern P11 takes each business process instance in the event log and generates the respective sequence of business activities that were found in events of the process instance. Only those business activities that are also present in the input code model will be represented in the business process view model.

Moreover, Pattern P12 (see Table 1) allows defining different branches in the business process view when two different process instances in the event model represent two sequences with a common and non-common part. For example, if there are two instances executing the sequences of activities $\{A, B, x, C\}$ and $\{A, B, y, C\}$, the discovered business process would be defined as the sequence of activities A and B, a branching sequence with activities "x" and "y" in parallel, and finally the common activity C. The different part of a sequence of activities (e.g. activities "x" and "y") are separated by means of a gateway element in the business process model (see Table 2).

3.5. Model transformation implementation

The proposed transformation [see Fig. 1(c)] is implemented using QVT (Query/View/Transformation).³¹ QVT consists of two different, but related, languages: *Operational Mappings* and *Relation*. It particularly uses *QVT Relation*, which provides a declarative and rule-based specification, since it facilitates the definition of the proposed patterns and declarative constraints that must be satisfied by the metaclass instances of the input and output models.

A model transformation defined through *QVT Relation* language consists of a set of relations with two kinds of domains, which define variables that can be matched

Table 2. Transformation between input and output elements through proposed patterns.

	Business Process Models									
	Task	Sequence Flow	Association	Data Object	Intermediate Event	Exception Event	Process	Start Event	End Event	Gateway
Code Models										
Callable unit	P2, P4, P5, P6, P9, P10									
Calls		P2, P4, P9, P10								
Reads			P5							
Writes			P6							
Storable Unit				P5, P6						
True flow					P9					
Flow						P10				
Event Models										
Event model							P11, P12			
Event resource								P11	P11	
Event	P11, P12	P11, P12								P12

in the type of a given model metaclass.³¹ Input domains (tagged with the *checkonly* keyword) define a set of metaclasses of the input metamodel and a set of constraint related to those metaclasses which must be fulfilled. Outputs domains (tagged with the *enforce* keyword) define a template of metaclasses and their properties that must be modified or created in the output model to satisfy the relation. The implementation consists of a set of relations by supporting all the proposed patterns which check the existence of the input elements of Table 2 (*checkonly* domains) and generates the output elements summarized in Table 2 (*enforce* domains).

To illustrate the implementation of the transformation Fig. 4 shows the “*ProcessInstance2Sequence*” relation, which implements Pattern P11 (see Table 1). Due to space limitation, this paper only shows this relation as example although the entire transformation is online available.³² This relation has three *checkonly* domains (lines 5 to 16) that are respectively defined on instances of the *ProcessInstance*, *Event* and *Process* metaclass. This relation is invoked from a previous relation that takes into account a particular code model and provides the actual values (as parameters) for these three *checkonly* domains. In summary, these domains check the existence of two different events in a same process instance. The *enforce* domain (see lines 17 to 37 in Fig. 4) creates the two respective business tasks (lines 22 and 26) and a sequence flow between those tasks (line 30). These elements are created in the business process view model only for contiguous business tasks, i.e. tasks that were executed, at least once, in a row. This constraint is evaluated in the *when* clause by invoking the query “*subsequent*” (see lines 42 to 45 in Fig. 4).

The final result obtained by the model transformation is a set of business process views with which to solve the problem of concept location identification. The next section provides a case study in order to demonstrate the feasibility of the proposal.

4. Case Study

In order to evaluate the proposal, this section provides a multi-case study with *AELG-members* — an author management system — and *V-Lab* — a web application of a chemical laboratory. The obtained results are compared with the results obtained in two previous case studies^{33,13} conducted with the same LIS by applying static and dynamic approaches in isolation without considering business process views in a combined way.

Despite the comparison with one more approach from the literature would be very appropriate to strengthen conclusions, such approaches are hardly ever validated, and other proposals were manually validated with a tool and a system under study non-available to replicate the study under the same conditions.

The case study was rigorously planned and executed by following the protocol for conducting case studies in the software engineering field proposed by Runeson *et al.*³⁴ The following subsections present in detail all the different case study stages according to this well-proven protocol (i.e. design, case selection, execution, data collection, analysis and interpretation, and threats to the validity).

```

1  relation ProcessInstance2Sequence {
2    xTaskName : String;
3    xTask2Name : String;
4    xProcessName : String;
5    checkonly domain eventModel instance : event::EventResource {
6    };
7    checkonly domain eventModel ev1 : event::Event {
8      name = xTaskName
9    };
10   checkonly domain eventModel em : event::EventResource{
11     name = xProcessName,
12     eventElement = inst : event::EventResource {
13       name = instance.name,
14       eventElement = ev2 : event::Event { name = xTask2Name }
15     }
16   };
17   enforce domain bpmn bpd: bpmn::BusinessProcessDiagram {
18     Pools = p :bpmn::Pool {
19       name = xProcessName,
20       ProcessRef = pr :bpmn::Process {
21         Name = em.name,
22         GraphicalElements = t1 :bpmn::Task {
23           Name = ev.name.substringAfter('.'),
24           process = parent : bpmn::Process { Name = em.name }
25         },
26         GraphicalElements = t2 :bpmn::Task {
27           Name = ev2.name.substringAfter('.'),
28           process = parent :bpmn::Process { Name = em.name }
29         },
30         GraphicalElements = secf1 :bpmn::SequenceFlow {
31           SourceRef = t1,
32           TargetRef = t2,
33           process = parent :bpmn::Process { Name = em.name }
34         }
35       }
36     };
37   };
38   when {
39     subsequent(instance, ev1, ev2)
40   }
41 }
42 query subsequent (proInst:EventResource, ev1:Event, ev2:Event):Boolean{
43   proInst.eventElement.indexOf(ev1)=1+proInst.eventElement.indexOf(ev2)
44   and event1 <> event2 and event1.name <> event2.name
45 }

```

Fig. 4. The “ProcessInstance2Sequence” QVT relation.

4.1. Design

The object of study is the proposed approach, and the purpose of this study is the evaluation of its effectiveness. The main hypothesis is that the discovery of business process views from code and event model in combination is suitable for achieve concept location between source code and executed business activities. The main research question is therefore defined as RQ.

RQ. *Can the approach obtain specific business process views located in code with better quality levels than the entire business process obtained with previous techniques?*

4.1.1. Independent variables

In order to answer the research question, the study considers as the independent variable the set of retrieved models for three kinds of approaches: static, dynamic and combined. Due to the fact that each model retrieved is considered as different analysis units, the study follows an *embedded* design according to the *Yin's* categorization.³⁵ In addition, the study can be considered as a multi-case study since it is conducted with two different LIS.

4.1.2. Dependent variables

Moreover, the study considers some measures as dependent variables for business process models obtained using each approach (i.e. static, dynamic and combined). These measures facilitate a quantitative analysis to answer the research question. The study particularly defines five measures:

- Relative size of the process view, which can only be evaluated for the combined approach (i.e. the percentage of the number of business activities present in the process view).
- Complexity of the business process model which is based on the structural intricacy³⁶ and is evaluated regarding the amount of sequence flows (arcs) with respect to the number of tasks (nodes) (1). Complexity is inversely related to understandability, since a very complex model makes it probably less understandable than simple models.
- Precision (2) that represents the exactness or fidelity of business process model retrieved.
- Recall (3) which is a measure of completeness regarding business process model retrieved.
- F-measure (4) which combines precision and recall by means of a harmonic mean.³⁷ It is due to the fact, that precision and recall have an inverse relationship and both measures cannot therefore be evaluated in isolation.

$$Complexity = \frac{\{sequence\ flows\}}{\{business\ tasks\}}, \quad (1)$$

$$Precision = \frac{\{recovered\ relevant\ tasks\}}{\{recovered\ relevant\ tasks\} + \{recovered\ non\ relevant\ tasks\}}, \quad (2)$$

$$Recall = \frac{\{recovered\ relevant\ tasks\}}{\{recovered\ relevant\ tasks\} + \{non\ recovered\ relevant\ tasks\}}, \quad (3)$$

$$F\ measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}. \quad (4)$$

Precision and recall measure the similarity between a mined business process M and a reference business process R . Precision indicates what proportion of M

matches R (i.e. how exact M is), while recall indicates what proportion of R is present in M (i.e. how complete M is). The study considers the *task* element as the *score unit* in order to apply these measures in a business process recovery scenario. Additionally, other elements such as sequence flows are implicitly considered by checking pre- and post-sequence flows of a certain task. It uses the opinions of business experts to discover which recovered tasks are or are not relevant. The precision measure (2) is therefore defined as the number of recovered relevant tasks divided by the total number of recovered tasks, while the recall measure (3) is defined as the number of recovered relevant tasks divided by the total number of relevant tasks.

4.2. Case selection

The first selected case is *AELG-members*, which is an existing author management system of a Spanish public administration. AELG is the Spanish abbreviation for Association of Writers in Galician Language. The system automates several services like author registration, cancelation of memberships and payment of fees. *AELG-members* was released four years ago, and it has had three medium modifications and a large modification (versions 1.1, 2.0, 2.1, and 2.2), thus, it is actually a LIS. From a technological point of view, *AELG-members* is a *Java* standalone application whose architecture follows the traditional structure into three layers: (i) the *domain* layer supporting all the business entities and controllers; (ii) the *presentation* layer dealing with the user interfaces; and (iii) the *persistency* layer handling data access. The total size of the legacy system is 23.5 KLOC (thousands of lines of source code).

The second system under study is V-Lab, which is a web application used by *Villasante Laboratory* a company in the water and waste industry. V-Lab manages information related to chemical laboratories, customers and products such as chemical analysis, dilutions, and chemical calibrations. The analyses supported by the application examined different parameters, including a large number of physical, chemical and microbiological parameters according to current regulations and laws for controlling water quality. From a technological viewpoint *V-Lab* is a *Java*-based web application following the MVC (Model-View-Controller) architecture. It was released six years ago, and it has undergone three major modifications with seven medium modifications in total. The version history without minor modifications was: 1.0; 1.1; 2.0; 2.1; 2.2; 3.0; 3.1; 3.2; 3.3; 3.4; 4.0. The total size of the legacy system is 28.8 KLOC.

In order to illustrate how the proposal works, the case study focuses on artifacts obtained during the application of the model transformation to the first system. In this sense, Fig. 5 shows the business process supported by *AELG-Members*, which is considered as the reference business process model. The main business activities carried out by the writers' organization, including, among others, "insert new author", "edit author", "establish fees", "import author" from different sources, and "print reports".

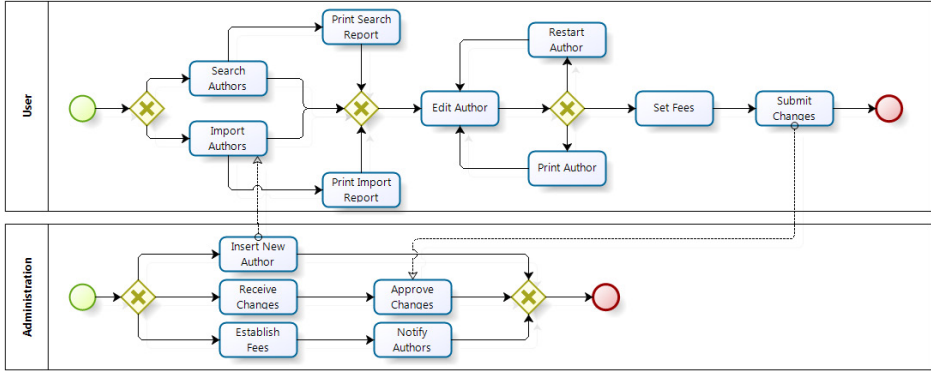


Fig. 5. Reference business process model of the *AELG-members* system.

4.3. Execution

The execution procedure consists of the following steps: (i) at the beginning, a set of code models is obtained for each package of legacy source code; (ii) in parallel, an event model is obtained by executing an instrumented version of the LIS; (iii) After that, in order to answer the research question, a set of business process views is obtained by combining the event model and each code model; (iv) finally, business process views are qualitatively analyzed and some measures are taken.

Figure 6 presents as example the three models involved in the proposed transformation with *AELG-Members*. First, the code model, which is automatically obtained through the mentioned static analysis technique (see e.g. Sec. 3.2), depicts all the callable units and the calls between them that are within the code package “*fees*” [see Fig. 6(a)]. Moreover, the event model obtained following the dynamic approach (see e.g. Sec. 3.2.2) contains events regarding the entire system [see Fig. 6(b)]. The event model shows the process “*Categories Management*” and some process instances in an expanded way.

Finally, Fig. 6(c) provides the business process view obtained by executing the proposed model transformation. The business process view represents the part of the “*Categories Management*” business process that is supported by the “*fees*” code package. The business process view in Fig. 6(c) exposes the three activities that are present in code and event model at the same time (i.e. `updateFeeType`, `getFeeType` and `addFeeType`). As a result, these three business activities (the concepts) of the business process view are located in the respective “*fees*” code package.

4.4. Data collection

After transforming all the code packages, relevant information related to the aforementioned measures is collected to be analyzed later. Table 3 provides the mean values for the measures depicted in Sec. 4.1, which were obtained for the combined approach as well as the static and dynamic one. The information regarding static

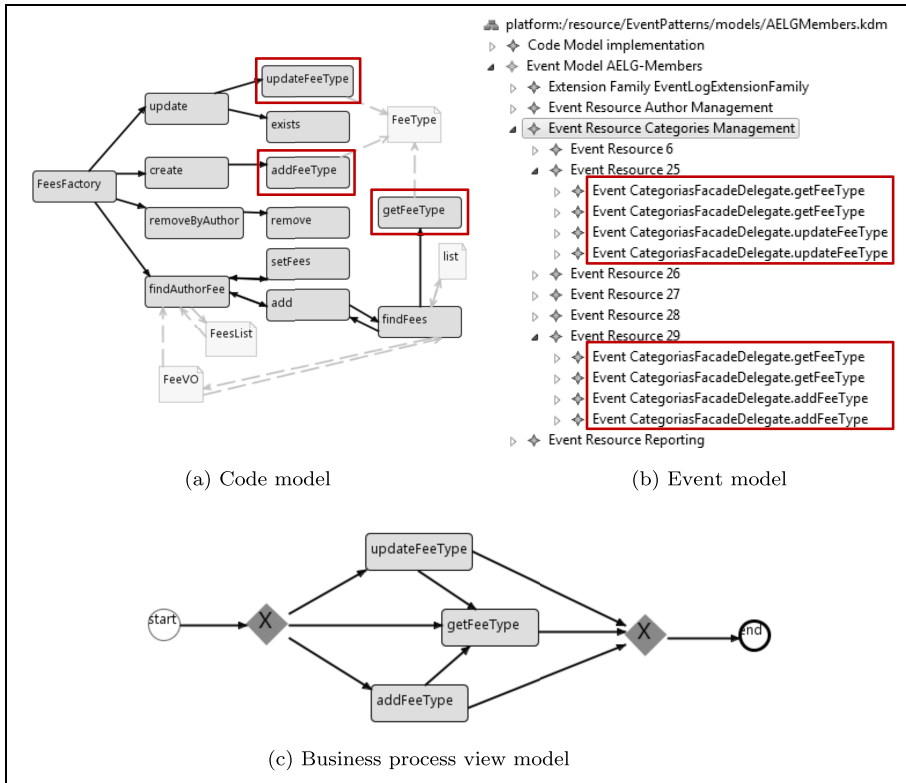


Fig. 6. The example transformation’s models for a particular source code package.

and dynamic approaches is taken from a previous case study involving the same systems.^{33,13}

Mean values of the collected data from AELG-Members corresponds to a sample size of 25 business process models for combined and static approach, and three for the dynamic one. In case of V-Lab, mean values were obtained from 32 business process models for combined and static approach, and four models for the dynamic one. While static and combined approach respectively generate a business process and a business process view for each source code package, the dynamic one directly obtains the three whole business processes from the execution of different business process instances in the event model. This is due to the fact that such models were discovered by means of process mining algorithms. These algorithms provide a few alternatives from the runtime information since it only provides the real sequence but not all the alternatives such as the static approach.

4.5. Analysis and interpretation

According to the main research question (see e.g. Sec. 4.1), the means of relative size are 21% and 43% (see Fig. 6) which means that the size of the business process

Table 3. Qualitative measurement for the combined, static and dynamic approach.

	Combined Approach			Static Approach			Dynamic Approach			
	Relative Size	Complexity	Precision Recall	F-Measure	Complexity	Precision Recall	F-Measure	Complexity	Precision Recall	F-Measure
<i>AEIG</i>										
<i>Mean</i>	0.21	1.34	0.81	0.72	1.82	0.51	0.78	0.61	1.51	0.76
<i>Std. Dev.</i>	0.13	0.28	0.04	0.02	1.35	0.07	0.11	0.07	0.87	0.10
<i>V-Lab</i>										
<i>Mean</i>	0.43	1.30	0.81	0.72	1.68	0.53	0.83	0.65	1.49	0.79
<i>Std. Dev.</i>	0.06	0.33	0.04	0.02	1.17	0.16	0.08	0.14	0.70	0.04

views retrieved for each source code package represent on average 21% of the entire business process supported by *AELG-Members* and 43% for *V-Lab*. Business process views generally are not disjoint sets of activities, i.e. they could be overlapping through some business activities repeated in different views. The advantage of smaller business process views regarding whole business processes is that it allows maintainers focusing on the particular parts of the business processes that are supported by a specific piece of source code. Moreover, the complexity ratio of business process views of the combined approach (1.34) is lower than models obtained using the static (1.82) or dynamic (1.51) approach in case of *AELG-Members* and a similar complexity ratio for *V-Lab* in case of combined approach (1.30) as well as for the static and dynamic approach (1.68 and 1.49 respectively) (see Fig. 6). Since understandability and complexity have an inverse relationship,³⁸ the understandability of business process views of the combined approach is higher than the understandability of business process models obtained by code models or event models in isolation. The effectiveness of the combined approach in both systems is therefore better than the previous approaches in isolation.

Regarding the measures related to the effectiveness of the technique (i.e. precision, recall and F-measure) the combined approach presents better results for both systems (see Fig. 6). The precision values obtained for business process views are better since these views discard some wrong business activities, which are recovered by using both static and dynamic approaches. Recall values are, however, a bit lower than values obtained for the other approaches. This is due to the fact that some business activities could not be considered in the event model as well as in the code model at the same time. As a result, these activities would not be recovered in business process views using the proposed combined approach. Anyway, the mean of F-measure for *AELG-Members*, which is used as a combined value of both measures, is higher (0.77) than F-measure mean of the static (0.61) and dynamic (0.72) approach. In case of *V-Lab*, the F-measure is the same (0.77), which is better than values obtained for the static (0.65) and dynamic (0.74) approach (see Fig. 6).

The quantitative analysis shows that there is not a great difference (0.05 and 0.03 for both systems) between the dynamic approach and combined one. Anyway, a qualitative analysis demonstrates that the combined approach is able to provide business process views (the main goal) with at least a similar F-measure, i.e. the efficiency is not altered. As conclusion, the main research question can therefore be answered as true.

4.6. Threats to the validity

Finally, the validity of the results must be assessed as unbiased and true for the whole population for which we wish to generalize the results. For this purpose, this section presents the threats to the validity of this study and possible actions to mitigate them.

The major threat to the *internal validity* concerns the particular implementation of the model transformation to obtain business process views. The particular implementation using QVT relations may affect the results (efficiency in particular) since the values measured might have been different if the proposed concept location technique is implemented with a different model transformation language. To mitigate this threat, the study could be replicated using several implementations according to different model transformation languages. Another threat to the internal validity is the set of workflow patterns proposed since additional patterns could be added to extend the transformation, or even the current patterns could be improved to consider additional information and aspects from the source code. In particular, these patterns do not consider explicitly the discovery of business activities from several callable units since it obtains compound activities from several fine-grained tasks previously obtained from atomic callable units. In order to mitigate this threat some clustering and refactoring techniques may be provided to compound business activities from atomic ones.

Regarding the *construct validity*, the main threat is the way in which the measured values are used to determine the suitability of the proposed approach. Future replications could be improved by considering the maintainers' opinion through some questionnaires in order to know if the concept location is enhanced thorough the obtained business process views.

Finally, *external validity* is concerned with the generalization of the results to the whole population, which is considered as LISs. The outgoing results could be generalized to this population. However, the specific cases based on the Java platform (which is additionally object-oriented) is a bias to be noted. Thus, the results can be strictly extended to this kind of LISs. In order to mitigate this threat, the study should be replicated involving other technologies. Furthermore, the approach does not have to be limited to LISs, since it could also be applied to new information systems, for which maintainers also want to understand the mappings between code fragments and business process activities.

5. Conclusions and Future Work

This paper presents a concept location approach based on the extraction of business process views, i.e. it considers business activities as the concept to be located. The approach, which follows the model-driven development principles, combines code models and event models to obtain business process views. Each view represents the part of the full business process that is supported by a particular piece of source code. As a result, business activities are mapped to the existing source code.

Compared to other proposals, this approach can improve the concept location during software modernization, since it uses business activities (at higher abstraction level) as concepts and links concepts and source code in two directions. The achieved concept location entails valuable knowledge when a LIS is modernized. First, when a subsystem is modernized, maintainers may know the business process

fragments, i.e. all the functionalities supported by the subsystem. Second, if a part of a business process is modified by an organization to adapt to environmental changes, maintainers may also know the specific pieces of source code involved in those business activities, which should be respectively modernized to align the LIS. Furthermore, business processes are supported in modern organizations by several heterogeneous IT systems (e.g. there are PAISs and traditional LISs at the same time). In this scenario, our approach provides good heterogeneity support since it uses standard representations. First, the approach uses event log models using the MXML format and obtained independently from PAISs or traditional LISs. Second, the approach uses code models represented using the recent standard KDM³⁰ to be used during process view generation.

The feasibility of the proposal has been assessed by means of a case study involving two real-life LISs, which were the cases under study in the validation of previous, precursory approaches. The study demonstrates that (i) the approach can obtain business process views from the combination of code and event models, and (ii) the obtained views have better quality levels than the full business processes, since some undesirable business activities are discarded in the business process views.

The work-in-progress focuses on the definition of additional patterns to recognize additional aspects of business processes views like parallel branches from asynchronous calls or compound complex business activities from atomic callable units. In addition, the study may be repeated with different LISs.

Acknowledgments

This work was supported by the FPU Spanish Program and the R&D projects PEGASO/MAGO (TIN2009-13718-C02-01) and GEODAS-BC project (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01). Additionally, this work was supported by the University of Innsbruck.

References

1. W. M. Ulrich, *Legacy Systems: Transformation Strategies* (Prentice Hall, 2002), p. 448.
2. J. Jeston, J. Nelis and T. Davenport, *Business Process Management: Practical Guidelines to Successful Implementations*, 2nd edn. (Butterworth-Heinemann (Elsevier Ltd.), USA, 2008), p. 468.
3. B. Paradauskas and A. Laurikaitis, Business knowledge extraction from legacy information systems, *J. Inf. Technol. Contr.* **35**(3) (2006) 214–221.
4. W.-J. V. D. Heuvel, *Aligning Modern Business Processes and Legacy Systems: A Component-Based Perspective (Cooperative Information Systems)* (The MIT Press, 2006).
5. W. van der Aalst and A. J. M. M. Weijters, Process mining, in *Process-Aware Information Systems: Bridging People and Software Through Process Technology*, eds.

- M. Dumas, W. van der Aalst and A. Ter Hofstede (John Wiley & Sons, Inc., 2005), pp. 235–255.
6. M. Castellanos, K. A. D. Medeiros, J. Mendling, B. Weber and A. J. M. M. Weijters, Business process intelligence, in *Handbook of Research on Business Process Modeling*, eds. J. J. Cardoso and W. M. P. van der Aalst, 2009 (Idea Group Inc., 2009), pp. 456–480.
 7. M. Dumas, W. van der Aalst and A. Ter Hofstede, *Process-Aware Information Systems: Bridging People and Software Through Process Technology* (John Wiley & Sons, Inc., 2005).
 8. W. M. P. van der Aalst, B. F. van Dongen, C. Günther, A. Rozinat, H. M. W. Verbeek and A. J. M. M. Weijters, ProM: The process mining toolkit, in *7th Int. Conf. Business Process Management (BPM'09) — Demonstration Track*, Springer-Verlag, Ulm, Germany, 2009, pp. 1–4.
 9. R. Pérez-Castillo, B. Weber, I. García Rodríguez de Guzmán and M. Piattini, Toward obtaining event logs from legacy code, *Business Process Management Workshops (BPI'10)*, 2010, Lecture Notes in Business Information Processing (LNBIP 66 - Part 2), pp. 201–207.
 10. R. Pérez-Castillo, B. Weber, I. García Rodríguez de Guzmán and M. Piattini, Process mining through dynamic analysis for modernizing legacy systems, *IET Softw. J.* **5**(3) (2011) 304–319.
 11. T. Eisenbarth, R. Koschke and D. Simon, Locating features in source code, *IEEE Trans. Softw. Eng.* **29**(3) (2003) 210–224.
 12. R. Pérez-Castillo, I. García-Rodríguez de Guzmán, O. Ávila-García and M. Piattini, On the use of patterns to recover business processes, in *25th Annual ACM Symp. Applied Computing (SAC'10)*, ACM, Sierre, Switzerland, 2010, pp. 165–166.
 13. R. Pérez-Castillo, B. Weber, I. García Rodríguez de Guzmán, M. Piattini and Á. S. Places, An empirical comparison of static and dynamic business process mining, in *26th Annual ACM Symp. Applied Computing (SAC'11)*, ACM, TaiChung, Taiwan, 2011, pp. 269–276.
 14. R. Eshuis and P. Grefen, Constructing customized process views, *Data Knowl. Eng.* **64**(2) (2008) 419–438.
 15. D. Schumm, F. Leymann and A. Streule, Process viewing patterns, in *14th IEEE Int. Enterprise Distributed Object Computing Conf. (EDOC)*, 2010.
 16. H. R. Motahari-Nezhad, R. Saint-Paul, F. Casati and B. Benatallah, Event correlation for process discovery From web service interaction logs, *VLDB J.* **20**(3) (2011) 417–444.
 17. W. van der Aalst, T. Weijters and L. Maruster, Workflow mining: Discovering process models from event logs, *IEEE Trans. Knowl. Data Eng.* **16**(9) (2004) 1128–1142.
 18. A. K. Medeiros, A. J. Weijters and W. M. Aalst, Genetic process mining: An experimental evaluation, *Data Min. Knowl. Discov.* **14**(2) (2007) 245–304.
 19. J. E. Ingvaldsen and J. A. Gulla, Preprocessing support for large scale process mining of SAP transactions, *Business Process Intelligence Workshop (BPI'07)*, LNCS 4928, 2008, pp. 30–41.
 20. C. W. Günthe and W. M. P. van der Aalst, A generic import framework for process event logs, *Business Process Intelligence Workshop (BPI'06)*, LNCS 4103, 2007, pp. 81–92.
 21. Y. Zou and M. Hung, An approach for extracting workflows from e-commerce applications, in *Proc. Fourteenth Int. Conf. Program Comprehension*, IEEE Computer Society, 2006, pp. 127–136.

22. Z. Cai, X. Yang and W. Wang, Business process recovery for system maintenance — An empirical approach, in *25th Int. Conf. Software Maintenance (ICSM'09)*, IEEE Computer Society, Edmonton, Alberta, Canada, 2009, pp. 399–402.
23. C. di Francescomarino, A. Marchetto and P. Tonella, Reverse engineering of business processes exposed as web applications, in *13th European Conf. Software Maintenance and Reengineering (CSMR'09)*, IEEE Computer Society, Fraunhofer IESE, Kaiserslautern, Germany, 2009, pp. 139–148.
24. N. Wilde and M. C. Scully, Software reconnaissance: Mapping program features to code, *J. Softw. Maint.* **7**(1) (1995) 49–62.
25. A. Marcus, A. Sergeev, V. Rajlich and J. I. Maletic, An information retrieval approach to concept location in source code, in *Proc. 11th Working Conf. Reverse Engineering*, IEEE Computer Society, 2004, pp. 214–223.
26. K. Chen and V. Rajlich, Case study of feature location using dependence graph, in *Proc. 8th Int. Workshop on Program Comprehension*, IEEE Computer Society, 2000, p. 241.
27. OMG, Business Process Model and Notation (BPMN) 2.0. Object Management Group, 2009, p. 496.
28. X. Zhao, C. Liu, W. Sadiq, M. Kowalkiewicz and S. Yongchareon, Implementing Process views in the web service environment, *World Wide Web* **14**(1) (2011) 27–52.
29. OMG, ADM Task Force by OMG. 2007 9/06/2009 [cited 2008 15/06/2009], <http://www.omg.org/>.
30. ISO/IEC, ISO/IEC DIS 19506. Knowledge Discovery Meta-model (KDM), v1.1 (Architecture-Driven Modernization). http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?ics1=35&ics2=080&ics3=&csnumber=32625. 2009, ISO/IEC, p. 302.
31. OMG, QVT. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, <http://www.omg.org/spec/QVT/1.0/PDF>. 2008, OMG.
32. R. Pérez-Castillo, MXML to KDM Transformation implemented in QVT Relations. 2011 29/03/2011 [cited 2011 29/03/2011], <http://alarcos.esi.uclm.es/per/rpdel-castillo/modeltransformations/MXML2KDM.htm>.
33. R. Pérez-Castillo, I. G.-R. de Guzmán and M. Piattini, Business process archeology using MARBLE, *Inf. Softw. Technol.* **53** (2011) 1023–1044.
34. P. Runeson and M. Höst, Guidelines for conducting and reporting case study research in software engineering, *Empirical Softw. Eng.* **14**(2) (2009) 131–164.
35. R. K. Yin, *Case Study Research, Design and Methods*, 3rd edn. (Sage, London, 2003).
36. E. Rolón, F. Ruiz, F. García and M. Piattini, Evaluation measures for business process models, in *21th ACM Symp. Applied Computing, Track on Organizational Engineering (SAC-OE'06)*, ACM, Dijon, France, 2006, pp. 1567–1568.
37. C. J. van Rijsbergen, *Information Retrieval*, 2nd edn. (Butterworths, London, 1979), p. 208.
38. H. A. Reijers and J. Mendling, A study into the factors that influence the understandability of business process models, *IEEE Trans. Syst. Man Cybern. A, Syst. Humans* **99** (2010) 1–14.