Integrating Event Logs into KDM Repositories

Ricardo Pérez-Castillo, Ignacio García-Rodríguez de Guzmán and Mario Piattini
University of Castilla-La Mancha
Paseo de la Universidad 4 13071
Ciudad Real, Spain
+34926295300
{ricardo.pdelcastillo, ignacio.grodriguez, mario.piattini}@uclm.es

Barbara Weber
University of Innsbruck
Technikerstraße 21a, 6020
Innsbruck, Austria
+435125076113
barbara.weber@uibk.ac.at

ABSTRACT
Business knowledge embedded in legacy information systems is a valuable asset that must be recovered and preserved when these systems are modernized. Event logs register the execution of business activities supported by existing information systems, thus they entail a key artifact to be used for recovering the actual business processes. There exists a wide variety of techniques to discover business processes by reversing event logs. Unfortunately, event logs are typically represented with particular notations such as Mining XML (MXML) rather than the recent software modernization standard Knowledge Discovery Metamodel (KDM). Process mining techniques consequently cannot be effectively reused within software modernization projects. This paper proposes an automatic technique to transform MXML event logs into event models to be integrated into KDM repositories. Its main implication is the exploitation of valuable event logs by well-proven software modernization techniques. The model transformation has been validated through a case study involving several benchmark event logs.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Restructuring, reverse engineering*, *and reengineering*. D.2.8 [**Software Engineering**]: Metrics – *Performance measures*. D.2.13 [**Software Engineering**]: Reusable Software – *reusable models*.

General Terms
Algorithms, Management, Measurement and Experimentation.

Keywords
Model-Driven Reengineering, Business Process, Event Logs, Knowledge Discovery Metamodel.

1. INTRODUCTION
Business processes are a key asset for companies since they represent the daily sequence of activities aimed at achieving their business objectives [20]. In addition, it has been demonstrated that companies which explicitly represent and manage their business processes get a better competitiveness level, since business processes management helps companies to adapt to environmental changes.

Most business processes are automatically supported in companies by means of their information systems [20]. These information systems undergo the phenomenon of software ageing and erosion over time as a result of uncontrolled maintenance, and they become Legacy Information Systems (LISs). When the maintainability degree of LISs diminishes under acceptable levels, these LISs must be replaced by other improved systems [2].

Re-implementing LISs from scratch is a possible solution to deal with the negative effects of the software erosion phenomenon. However, greenfield developments are not the most suitable solution, since it entails various economical and functional risks [17]. Firstly, the economic aspect of companies is affected, since the replacement of an entire LIS, by implementing a new system from scratch, can imply a low Return of Investment (ROI) with regard to the old system. In addition, the development or purchase of the new system might exceed a company's budget. Secondly, the new system may have a lack of specific functionalities that are missing in the new one. This is due to the fact that over time new meaningful business knowledge is incorporated into LISs during their maintenance. That business knowledge embedded in LISs, which is not present anywhere else, would be lost with new developments from scratch.

Software modernization is a more suitable solution to address the software erosion problem, since it saves cost and preserves the embedded business knowledge. Software modernization is the concept of evolving LISs with a focus on all aspects of the current system's architecture and the ability to transform current architectures into target architectures. Software modernization facilitates the preservation of business knowledge, since it tends to recover all the embedded knowledge without discarding such knowledge during the development of enhanced system [12].

Embedded business knowledge preservation entails two main challenges, which are in line with the current challenges in reverse engineering [3]. The first one is the *discovery* or elicitation of business knowledge itself and the second one is the *effective usage* of this knowledge to take advantage during evolution and modernization of LISs.

The first challenge (i.e., discovery of the embedded business knowledge) has been widely addressed in literature through some *business process mining* techniques and algorithms [18], which allow discovering the actual business processes carried out by companies. These techniques take as input event logs which represent the specific business activities executed by a system following the Mining XML (MXML) format [19].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'12, March 25-29, 2012, Riva del Garda, Italy.
Copyright 2012 ACM 978-1-4503-0857-1/12/03…$10.00.

The second challenge concerns the effective usage of the discovered business knowledge to accomplish more effective software modernization processes. The evolved information systems must support, or be aligned with, the actual business processes previously discovered [5].

This paper focuses on this second challenge, since the business process mining techniques so far focuses on discovering business processes from particular even logs. However, these reverse engineering techniques were not thought to produce valuable event logs to be used in whole modernization projects. This is owing to the fact that business process mining techniques use event logs following the MXML format, which cannot be used together with the recent specifications of the Architecture-Driven Modernization (ADM) initiative [8], which has been especially provided for modernizing LISs.

This paper addresses this problem from the perspective of providing a model transformation for reversing event logs into standard repositories according to the Knowledge Discovery Metamodel (KDM). KDM is the specification proposed by the ADM initiative to abstract and represent all the legacy software artifacts and viewpoints of LISs during the reverse engineering stage. The main advantage is that event logs transformed into KDM models can be integrated into ADM-based processes so that synergies between event models and the remaining kinds of models (e.g., code model, database model, etc.) can be exploited together and in a homogeneous and standardized way. For example, feature location techniques may be improved since features of event models (as well as the respective discovered business process models) can be easily mapped with source code elements in KDM models [7].

In order to demonstrate the feasibility and facilitate the adoption of our proposal, a model transformation has been implemented using QVTr (*Query / View / Transformation Relations*) and a case study involving a real life LIS has been conducted. The case study indicates that (i) the model transformation is effective, since it can obtain KDM models representing the information registered in event logs; and (ii) the transformation is efficient, since it can be executed in a linear time regarding the number of events.

The remainder of this paper is organized as follows: Section 2 explains the background of the paper to understand the context where the model transformation is applied. Section 3 provides the model transformation proposed. Section 4 describes the case study. Finally, Section 5 presents conclusions and future work.

2. BACKGROUND

This section introduces the key concepts to provide a better understanding of the proposal: business process mining and software modernization.

2.1 Business Process Mining

Business process mining describes a family of a-posteriori analysis techniques exploiting the information recorded in an event log [18]. Event logs sequentially record the business activities executed in a Process Aware Information Systems (PAIS) [18]. PAIS's are systems that explicitly manage the execution of business processes (e.g., Customer Relationship Management (CRM) systems or Enterprise Resource Planning (ERP) systems).

Business process mining allows companies to know what is actually occurring within their organizational environment, since their business process models could not be aligned with the real-life operation [4]. According to [18], there are three kinds of business process mining techniques: (i) **discovery** which deal with the construction of business processes when there is no a-priori business process model; (ii) **monitoring** which are used for run-time analysis such as checking if actual business processes are in line with a reference business process model; and (iii) **extensions** which extract and enrich business process models by using new aspects recovered from event logs. The proposal is used in the discovery type, since it aims to use event logs to discover business processes and carry out software modernization projects that preserve that business knowledge.

2.2 Software Modernization

Software modernization is an approach supporting evolutionary maintenance [7], i.e., it consists of understanding and evolving existing software artifacts and restores the value of LISs. Software modernization does not replace traditional reengineering, but it combine it together with new model-driven development principles [9]. This means that traditional reengineering is improved by treating all involved software artifacts as models and establishing automatic, reusable model transformations between models at different abstraction levels [12].

The Object Management Group (OMG) has launched the ADM initiative to standardize the software modernization approach [9]. The first specification defined within ADM is KDM, which is a metamodel to represent all the software artifacts of LISs. In addition, KDM has been recently recognized as the ISO/IEC 19506 standard [13].

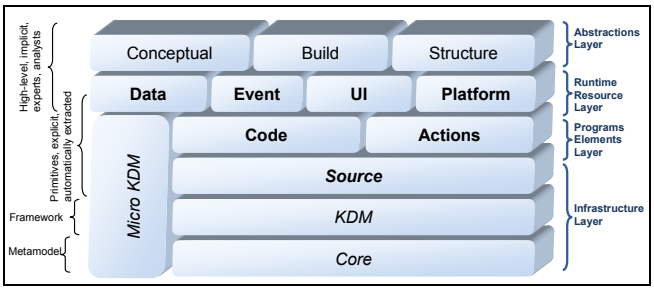

Figure 1. KDM Layers, packages, and concerns (adapted [7])

KDM groups knowledge about legacy software artifacts into various orthogonal concerns that are in turn organized in different abstraction layers, each based on the previous one (see Figure 1). The **infrastructure layer** is at the lowest abstraction level and defines a small set of concepts used systematically throughout the entire KDM specification. The **program elements layer** offers a broad set of metamodel elements to provide a language-independent intermediate representation for various constructs defined by common programming languages. This layer has two packages: *Code* and *Action*. The **runtime resource layer** represents resources managed by the legacy system's operation environment (i.e., it focuses on aspects that are not contained within the code itself). It has four packages: *Data*, *Event*, *UI* and *Platform*. The *Event* package is the portion of the metamodel used to represent the KDM event model from the MXML log models. Finally, the **abstraction layer** defines a set of meta-model elements to represent domain-specific knowledge and a business-overview of LISs.

The KDM specification facilitates the automation of the software modernization process, since it defines the items that a reverse

engineering tool should discover and a software analysis tool may use. In the past, traditional reverse engineering tools have been built as silos where each tool recovers and analyzes different proprietary content in a single silo. For example, maintainers would use a reverse engineering tool for the source code and another tool for the legacy database. At the end, maintainers would have two proprietary and independent models, a source code model and a database model, which must also be analyzed independently. Nowadays, KDM makes it possible to build reverse engineering tools in a KDM ecosystem where reverse engineering tools recover knowledge regarding different artifacts, and the outgoing knowledge is represented and managed in an integrated and standardized way through a KDM repository. Software analysis tools can therefore be plugged into the KDM repository to generate new meaningful knowledge in the same repository.

Transforming MXML logs to KDM event models allows taking advantage of the KDM ecosystem. The obtained KDM models can therefore be integrated into software modernization projects.

3. REVERSING EVENT LOGS

This section presents the technique proposed to obtain KDM models from event log models.

3.1 Involved Metamodels

This section presents the metamodels used to represent the input models (MXML event logs) and the output models (KDM event models).

3.1.1 The MXML Metamodel

Event logs are represented according to the MXML notation. MXML is the notation commonly used to represent event logs to be exploited in business process mining techniques [19].

Figure 2 provides the respective metamodel (using the *Ecore* metamodeling language as defined by Eclipse™ platform) for representing MXML event logs. The MXML metamodel represents an event log as an instance of the *WorkflowLog* metaclass. Each log consists of a set of instances of the *Process* metaclass that contains, in turn, several *ProcessInstance* elements. Each instance of the *ProcessInstance* metaclass represents a specific execution of a business process using particular data. For example, let us imagine a business process of a bank company. In this example, there could be different business process instances for different customers.

Each business process instance has a sequence of instances of the *AuditTrailEntry* metaclass (see Figure 2). The *AuditTrailEntry* metaclass represents events and consists of four main elements: (i) the *WorkflowModelElement* that represents the executed business activity; (ii) the *EventType* that represents if the activity was started (start) or was completed (complete); (iii) the *Originator* that provides the user who started or completed the business activity; and finally (iv) the *Timestamp* that records the date and time of the event.

Moreover, all the elements previously depicted can have one or more instance(s) of the *Data* metaclass. The *Data* metaclass groups several instances of the *Attribute* metaclass and aims to include additional, relevant information in different elements of the event log. This is the mechanism provided by the MXML metamodel to achieve the extension of the models conforming to this metamodel when it is necessary.

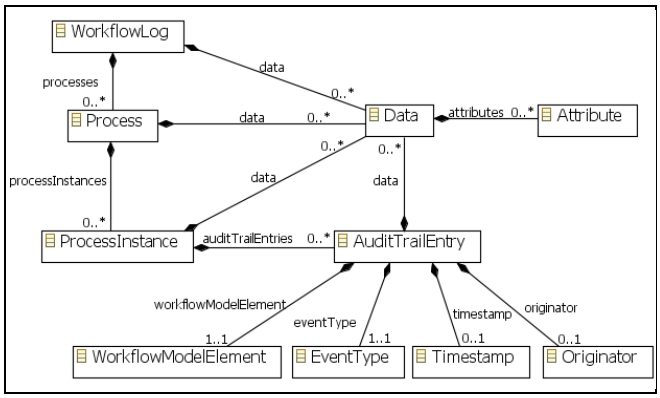

Figure 2. The MXML metamodel (using *Ecore*)

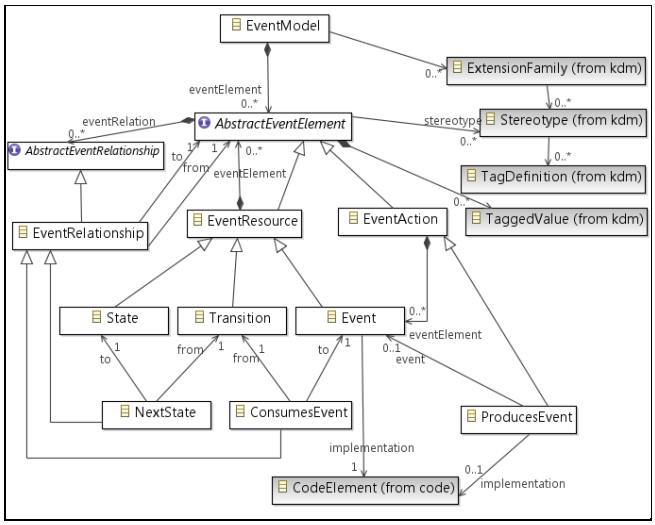

Figure 3. The KDM event metamodel and extensions

3.1.2 The KDM Event Metamodel

The KDM Event metamodel corresponds to the portion of the KDM metamodel depicted in the *Event* package (see Figure 1). This metamodel defines a set of metaclasses for representing states and state transitions caused by a set of events (see white metaclasses in Figure 3).

The KDM event metamodel defines the *EventModel* metaclass to depict an event model in KDM. Each instance of the *EventModel* metaclass aggregates a set of *EventResources* of a certain LIS. The *EventResource* metaclass is specialized into the *State*, *Transition* and *Event* metaclasses. The *Event* metaclass has two features: the *name* of the event and the *kind* feature depicting the type of the event (i.e., start or complete).

Each instance of the *EventResource* metaclass is related to one or more instances of the *AbstractEventRelationship* metaclass for representing additional relationships between events apart from transitions. The *AbstractEventRelationship* metaclass is specialized into two metaclasses: the *NextState* metaclass establishes a relationship from a *Transition* to the given *State* after this event-based transition; and the *ConsumeEvent* metaclass defines a relationship between a *Transition* and the *Event* that triggers that transition (see white metaclasses in Figure 3).

Table 1. Set of basic transformation rules

ID	Rule Description
R1	Each instance of the *WorkflowLog* metaclass is transformed into an instance of the *EventModel* metaclass in the output model
R2	Each instance of the *AuditTrailEntry* metaclass is transformed into an instance of the Event metaclass in the output model
R3	Instances of the *WorkflowModelElement* and *EventType* metaclass, belonging to an instance of the *AuditTrailEntry* metaclass, are respectively incorporated into the features 'name' and 'kind' of the respective instance of the *Event* metaclass (see R2).
R4	Instances of the *Attribute* metaclass with the name feature 'implementation' are transformed into instances of the *CodeElement* metaclass within the respective instance of the Event metaclass in the output model (see R2).

Moreover, the KDM event metamodel extends the KDM *action* package from a previous layer (see Figure 1) since it defines a set of event actions that can be used with the *EventResources* (see white metaclasses in Figure 3). These event actions can be represented by means of instances of the *ProducesEvent* metaclass which defines the event that is produced by a particular code element. The *ProducesEvent* metaclass together with the *Event* metaclass have the reference *implementation* that is defined as an instance of the *CodeElement* metaclass. Such references enables feature concept location when event logs are represented as an event model in KDM.

3.2 Transformation Rules

After the introduction of the involved metamodels, this section presents a set of rules defined for transforming MXML event log models into KDM event models. First of all, a KDM event model must be created from the MXML event log model (see R1 in Table 1). Moreover, events entail the key element of MXML event log models, thus, events must be transformed into the KDM event model (see R2 in Table 1).

Furthermore, the information concerning the four components of an event (i.e., business activity name, type, originator and timestamp) must be represented in the KDM event model. On the one hand, the name and type of the events in the MXML models are represented in the KDM event models by using the features of the *Event* metaclass respectively (see R3 in Table 1). On the other hand, the information concerning the originator and timestamp cannot be directly represented in the KDM model according to the KDM event package. For this reason, the KDM event model must be extended with additional metaclasses so that it can support this information. Besides originator and timestamp information, additional MXML elements such as *Process* and *ProcessInstance* elements exist which cannot be represented in output models according to the original KDM event metamodel (cf. Section 3.3).

Finally, all the events, which represent executed business activities, are mapped to their respective pieces of source code by means of R4 (see Table 1). This is possible since the event log is represented using the KDM metamodel and thus the KDM event model can be linked with other models (e.g., the KDM code model) according to other metamodel packages defined in the KDM specification (see Figure 1). This mapping between events and source code is a valuable knowledge during software modernization.

Table 2. Extended set of transformation rules

ID	Rule Description
R5	An instance of the *ExtensionFamily* metaclass is created for each instance of the *EventModel* metaclass in the output model (see R1 in Table 1). This instance contains four instances of the *Stereotype* metaclass. In turn, each *Stereotype* instance contains an instance of the *TagDefiniton* metaclass. The values of these four stereotypes are: <*process*>, <*processInstance*>, <*originator*> and <*timestamp*>.
R6	Each instance of the *Process* metaclass is transformed in the output model into an instance of the *EventResource* metaclass with an instance of the *TaggedValue* metaclass. The *tag* feature of this instance links to the <*Process*> stereotype, and the *value* feature represents process name.
R7	Each instance of the *ProcessInstance* metaclass is transformed in the output model into an instance of the *EventResource* metaclass with an instance of the *TaggedValue* metaclass. The *tag* feature of this instance links to the <*ProcessInstance*> stereotype, and the *value* feature represents the name of the business process instance.
R8	Instances of the *Originator* and *Timestamp* metaclass are transformed into two instances of the *TaggedValue* metaclass which are added to the respective instance of the *Event* metaclass (see R2 in Table 1). The instances of the *TaggedValue* metaclass respectively define their *tag* features as <*Originator*> and <*Timestamp*> stereotype, and their *value* feature with the name of the originator and timestamp registered in the input model.

3.3 Additional Rules and KDM Extension

In order to represent all the information registered in a MXML model in the KDM model, we propose an extension of the KDM event metamodel. Despite the KDM metamodel needs to be extended, the impact of this extension on well-proven and KDM-based tools is not problematic since it is carried out with the own extension mechanism of the KDM standard. Besides this fact, most elements of the event model are present in the core of KDM which is used for many tools.

The standard extension mechanism of KDM is the extension families. These are established by means of an instance of the *ExtensionFamily* metaclass, which can exist for any kind of KDM model, e.g., an instance of the *EventModel* metaclass. Each extension family defines a set of instances of the *Stereotype* metaclass which contain, in turn, a set of instances of the *TagDefinition* metaclass (see highlighted metaclasses in Figure 3). Stereotypes define a wide concern while tag definitions specify the new elements. Tag definitions established in the extension family are then used in standard elements of the KDM metamodel by means of instances of the *TaggedValue* metaclass (see highlighted metaclasses in Figure 3). Tagged values allow changing or adjusting the meaning of those elements by associating a value with a previously defined tag.

According to the extension mechanism, some additional rules are established. R5 refines R1 by adding the creation of the extension family within the event model (see Table 2). The extension family collects four stereotypes with a tag definition: <*process*>, <*processInstance*>, <*originator*> and <*timestamp*>. These stereotypes are used in different elements of the KDM event model. R6 and R7 respectively group business processes and

business process instances using the *EventResource* metaclass and a tagged value (see Table 2). Moreover, the originator and timestamp are represented by incorporating tagged values to the respective event in the output model (see R8 in Table 2).

```
1  top relation auditTrailEntry2Event {
2    xEventName : String;
3    xEventType : String;
4    xOriginatorName : String;
5    xDate : String;
6    xProcessInstanceName : String;
7    xProcessName : String;
8    xModelName : String;
9    checkonly domain mxml ate : mxml::AuditTrailEntry {
10     workflowModelElement = wme : mxml::WorkflowModelElement
11       name = xEventName
12     },
13     eventType = type : mxml::EventType {
14       type = xEventType
15     },
16     originator = originator : mxml::Originator {
17       name = xOriginatorName
18     },
19     timestamp = timestamp : mxml::Timestamp {
20       date = xDate
21     },
22     processInstance = pi : mxml::ProcessInstance {
23       name = xProcessInstanceName,
24       process = p : mxml::Process {
25         name = xProcessName,
26         workflowLog = wl : mxml::WorkflowLog {
27           name = xModelName
28         }
29       }
30     }
31   };
32   enforce domain event eventModel:event::EventModel{
33     name = xModelName,
34     eventElement = eRes:event::EventResource {
35       name = xProcessName,
36       eventElement = eRes2:event::EventResource {
37         name = xProcessInstanceName,
38         eventElement = event : event::Event {
39           name = xEventName,
40           kind = xEventType,
41           taggedValue = originatorTag : kdm::TaggedValue {
42             tag = ot : kdm::TagDefinition {
43               tag = 'Originator'
44             },
45             value = xOriginatorName
46           },
47           taggedValue = timestampTag : kdm::TaggedValue {
48             tag = dt : kdm::TagDefinition {
49               tag = 'Timestamp'
50             },
51             value = xDate
52           },
53           implementation = codeElement : code::CodeElement {
54             name = xEventName
55           }
56         }
57       }
58     }
59   };
60   when {
61     processInstance2eventResource (pi, eventModel);
62   }
63 }
```

Figure 4. The 'auditTrailEntry2Event' QVT relation.

3.4 Implementation

An executable version of the model transformation has been implemented using QVT (Query/View/Transformation) [10]. QVT consists of two different, but related, languages: *Operational Mappings* (QVTo) which provides an imperative and procedure-based specification; and *Relations* (QVTr) which provides a declarative and rule-based specification. We particularly use QVTr due to its declarative nature facilitating the definition of the proposed rules as well as declarative constraints that must be satisfied by the metaclass instances of the input and output model.

Due to the space limitation this paper only shows one relation as example. The full transformation, however, is available online [11]. Figure 4 shows the *'auditTrailEntry2Event'* relation illustrating the implementation of rules in QVT. In this relation, the *checkonly* domain (lines 9 to 31) is defined on instances of the *AuditTrailEntry* metaclass. This input domain checks the existence of the four elements of an event (i.e., the business activity, type, originator and timestamp). The input domain also evaluates the existence of the process instance, process and the event log where the *auditTrailEntry* element belongs (lines 22 to 30). On the other hand, the *enforce* domain (lines 32 to 59) creates an instance of the *Event* metaclass (line 38) according to Rule R2. This event is created within the respective log, process and process instance. The originator and timestamp are added with the appropriate stereotype according to the Rule R8 (lines 41 to 52). Finally, the *when* clause invokes the *'processInstance-2eventResource'* to check, as a pre-condition, that the respective process instance was previously created by means of the invoked relation.

4. CASE STUDY

This section validates the proposed model transformation by means of a case study that involves ten different event logs. The case study is based on the formal protocol proposed by *Runeson et al.* [16] for conducting case studies in the software engineering field, which allows to improve the validity and repeatability.

4.1 Design

The *object* of this study is the proposed model transformation, while the *purpose* is the evaluation of its effectiveness and efficiency. In order to evaluate these properties, two research questions are established:

RQ1 – Can the transformed KDM event models be integrated into KDM repositories?

RQ2 – Is the model transformation executed in a reasonable time?

The study follows a *multi-case* and *holistic* design. The study is *multi-case* because it evaluates ten different MXML models. In addition, the study is holistic since the model transformation is applied to each case as a whole, and does not consider several analysis sub-units. As a result, the independent variables of this study are the different input MXML models.

The first research question (RQ1) is evaluated by means of qualitative research. The qualitative evaluation focuses on the effectiveness of the proposed model transformation, which is measured by using the *indirect assessment* of the *external quality*. The *indirect assessment* [1] consists of the comparison of (aggregations of) metric values between input and output models. In addition, the study uses metrics concerning external quality since it represents the quality change induced on a model by the transformation, while internal quality metrics (e.g., understandability, modifiability, reusability, etc.) represent the quality of the transformation itself, which is not the objective of this study. The evaluation of RQ1 is aimed at knowing if obtained models are able to represent the respective event model within a KDM repository.

The second question (RQ2), in turn, is quantitatively evaluated. Therefore, some metrics are used to evaluate this question, which are considered as the dependent variables of the study. The evaluation of RQ2 uses the number of business process instances and the number of events of the input MXML models. The

efficiency of the model transformation is measured through the time spent on the transformation execution. The study uses the tool *Medini QVT* [6] as the execution environment since it supports the validation and execution of *QVTr* transformations. The computer used for the execution consists of a dual processor of 2.66 GHz and a RAM memory of 4.0 GB.

4.2 Case Selection

Cases under study cannot be randomly selected, but the cases must be selected according to some criteria in order to ensure the case would provide strengthened conclusions from their evaluation. For this reason, the case selection stage defines a criterion list to choose a suitable set of cases.

The set of cases are selected from the set of benchmark MXML models provided by the *Process Mining Group* that are available in [15] as well as other event logs registered by information systems of partner companies. The case selection procedure establishes the following criteria to select the most appropriate cases: C1 – The input models must have two or more processes and different process instances for each one, i.e., there must be events in different processes instances and process. This criterion avoids the evaluation of a sequence of events of a single process, which does not represent real-world logs. C2 – The entire set of logs must have logs with a different number of events. The objective of this criterion is to provide a set of logs with different size to show the behavior of the transformation for small, medium and large logs. C3 – Due to standards considered by the proposal, event logs must be MXML-compliant.

Table 3 shows the ten input models under study. The selected set of cases consists of MXML logs with an average size of 12474 events per log. The minimum size of a log is 66 events (for the model *'insuranceclaimexample'*) and the maximum size is 73799 (for the model 'grouped_t32f0n50'). In addition, the number of process instances varies between 6 and 1800 (see Table 3). According to the source of the models, on the one hand, seven models (M1 to M5, see Table 3) are selected from the set of benchmark MXML logs as used in other studies [15]. On the other hand, three models (M6 to M8, see Table 3) are selected from the event logs registered from traditional (non-process-aware) information systems of partner companies.

4.3 Case Study Procedure

Besides the case study design, a procedure must be defined to execute the study. The execution procedure consists of in the following steps:

1. The set of appropriate cases is selected according to the selection criteria. After case selection, each case is taken one by one to execute the proposed transformation, i.e., steps 2 to 4 are repeated for each case.
2. The MXML log is analyzed using *ProM* tool [19]. *ProM* is an extensible framework that supports a wide variety of process mining techniques in the form of plug-ins. Thanks to *ProM* the log can be analyzed to obtain, for example, the number of events or process instances. All this information is annotated to be evaluated later.
3. The MXML is then transformed to a KDM event model. The QVTr transformation is executed by means of an Eclipse™ plug-in that we have specially developed for this purpose using EMF/GMF, the Eclipse™ metamodeling framework. The plug-in embeds the open source QVTr engine of Medini QVT [6] in order to execute the proposed transformation. The plug-in additionally incorporates two editors and evaluators for the two kinds of models: MXML log models, and KDM models.
4. After the execution, the obtained KDM event model is analyzed according to the aforementioned metrics (cf. Section 4.1). All this information is annotated to be analyzed later.
5. When steps 2 to 4 are executed for all the cases under study, all the collection information (see Section 4.4) is analyzed to answer the research questions and draw conclusions (see Section 4.5).
6. Finally, the results of the study are reported through an appropriate discussion as well as an evaluation of the possible threats to the validity of the study (see Section 4.6).

Table 3. Data obtained during the case study conduction

Id	MXML Event Log	# Process Instances	# Events	Transform. Time (s)
M1	DutchRentalHouseOrganization	1000	13262	1947
M2	insuranceclaimexample	6	66	45
M3	logs_Afschriften	374	9174	1721
M4	outpatientclinicexample	1000	14000	2274
M5	grpd_al1	98	1794	3
M6	AELG-Members	63	856	11
M7	CHES	28	100	6
M8	Villasante Lab	149	546	10
	Mean	551.8	12474.3	687.4
	Std. Deviation	612.1	22297.9	917.4

4.4 Data Collection

Data to be collected during the conduction of the study as well as their sources have to be defined before the execution in order to ensure the repeatability of the study.

Table 3 shows the data collected during the execution of the model transformation, which provides: (i) the case identifier; (ii) the name of the MXML event log, (iii) the number of process instances and (iv) events; (v) the time spent on executing the transformation in seconds; and finally (vi) the last column indicates if the KDM event model was or was not obtained with any bug or mistake.

4.5 Analysis and Interpretation

After execution and data collection, the data is analyzed to obtain the chains of evidence to answer the established research questions.

In order to answer RQ1, the suitability of the obtained KDM event models to be integrated into KDM repositories is qualitatively evaluated. Although all the input models were analyzed, Figure 6 illustrates the qualitative analysis using model M6 as an example. Figure 6 (left) shows the KDM event model obtained for input model M6. The events of this model are obtained with information of the piece of source code that supports the business activity executed in an event. This information is stored in the 'implementation' feature of each *Event* element (see Figure 6 left). Those pieces of source code referenced from the KDM Event model are represented in a respective KDM Code model (see Figure 6 right) by means of different *CodeElements*. In the example KDM Code model, such pieces of source code were particularly represented as *MethodUnit* elements. The KDM Code model is obtained from the LIS by means of the mentioned technique based on static analysis, which was evaluated in a previous work [14]. In this way, all the KDM Event and Code models were therefore linked within the same KDM Repository. As a result, RQ1 can be answered as true, i.e., the proposed model

transformation can obtain KDM event models from MXML models and integrate them into a KDM repository.

Moreover, *RQ2* must be evaluated. The time spent on the execution varies between 3 seconds for case M5 and 2274 seconds (a bit more than half an hour) for the biggest case M4, which has 14000 events (see Table 3). The mean of the transformation time is 687.4 seconds (approximately 11 minutes). These values seem a feasible time for models with more than 1000 events. However, the scalability of the transformation must be demonstrated to ensure its applicability for larger models.

Our hypothesis is that the transformation time is linear regarding the size of input models, since the transformation has a theoretical algorithmic complexity *O(n)*, where *n* represent the number of events. In order to confirm this hypothesis Figure 5 provides a linear regression model, which considers the model size as the independent variable and the transformation time as dependent variable. The correlation coefficient R^2 of the regression model is 0.975 (see Figure 5). Since R^2 is very close to 1, our hypothesis is confirmed. As a consequence, the scalability of the proposed model transformation is ensured and RQ2 can be positively answered.

4.6 Threats to the Validity

The validity of the study must be evaluated by considering the possible threats and the list of actions to mitigate them. Firstly, the *internal validity* is threatened by the execution environment used to conduct the study, since the time values might vary. To mitigate this threat, the study could be repeated and compared by using different QVTr transformation engines apart from *Medini QVT*.

Moreover, the *external validity* is concerned with the generalization of the results to a whole population. Since the regression model is calculated with only eight values, the generalization could be improved by more replications.

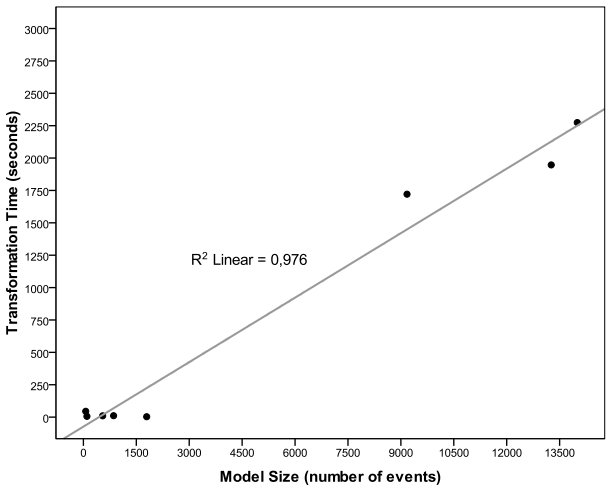

Figure 5. Linear regression for transformation time.

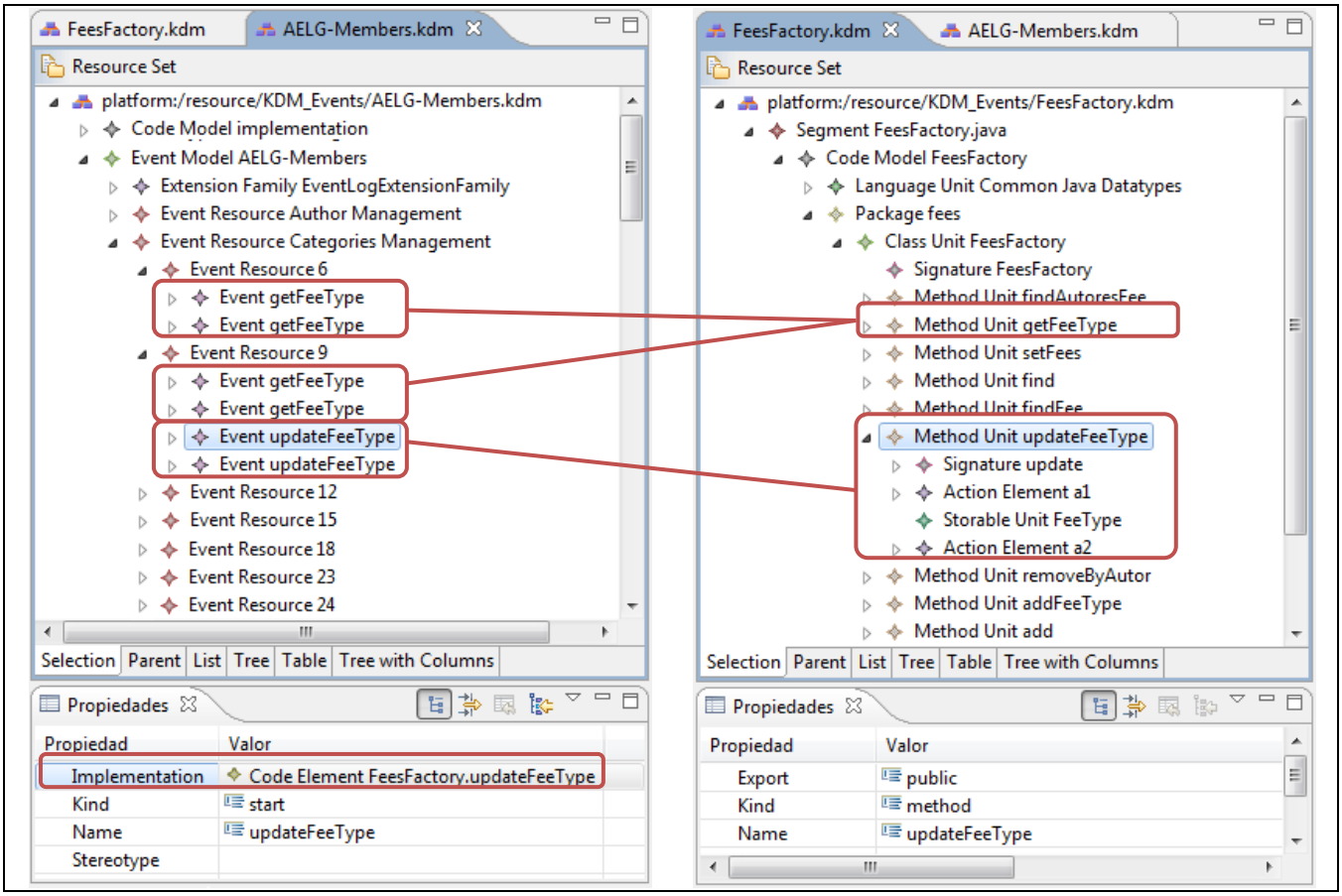

Figure 6. Integration of an event model (left) into a KDM repository together with a code model (right).

5. CONCLUSIONS AND FUTURE WORK

This paper has proposed an approach to transform event logs into KDM event models in order to integrate them into the KDM ecosystem. As a result, the KDM event models can be used in combination with other embedded knowledge recovered through reverse engineering. The main hypothesis is that the KDM ecosystem allows maintainers to obtain synergies from all the related knowledge to improve software modernization activities such as feature location among other.

This work provides an implementation of the model transformation using QVTr as well as a supporting tool in order to facilitate its validation and adoption by the industry. In fact, the transformation is validated through a case study involving ten event logs (seven benchmark logs and three logs obtained from the execution of real-world LISs). The case study shows that the proposed approach is able to obtain KDM event models from event logs, which can be integrated into a KDM repository. The main advantage of this fact is that KDM event models can be mapped to particular pieces of source code that supports the events. Moreover, the transformation time is linear, and therefore the approach could be used with larger event logs.

The future work will address the repeatability of the case study using additional and different event log models in order to deal with the detected threats and obtain strengthened conclusions.

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) and MOTERO (JCCM and FEDER, PEII11-0366-9449).

REFERENCES

[1] Amstel, M.F.v., The Right Tool for the Right Job: Assessing Model Transformation Quality, in 34th Annual Computer Software and Applications Conference Workshops. 2010, IEEE Computer Society: Seoul, Korea. p. 69-74.

[2] Bianchi, A., D. Caivano, V. Marengo, and G. Visaggio, Iterative Reengineering of Legacy Systems. IEEE Trans. Softw. Eng., 2003. 29(3): p. 225-241.

[3] Canfora, G., M.D. Penta, and L. Cerulo, Achievements and challenges in software reverse engineering. Commun. ACM, 2011. 54(4): p. 142-151.

[4] Castellanos, M., K.A.d. Medeiros, J. Mendling, B. Weber, and A.J.M.M. Weitjers, Business Process Intelligence, in Handbook of Research on Business Process Modeling, J. J. Cardoso and W.M.P. van der Aalst, Editors. 2009, Idea Group Inc. p. 456-480.

[5] Heuvel, W.-J.v.d., Aligning Modern Business Processes and Legacy Systems: A Component-Based Perspective (Cooperative Information Systems). 2006: The MIT Press.

[6] ikv++, Medini QVT. http://www.ikv.de/index.php?option=com_content&task=view& id=75&Itemid=77. 2008, ikv++ technologies ag.

[7] ISO/IEC, ISO/IEC 14764:2006. Software Engineering -- Software Life Cycle Processes -- Maintenance. http://www.iso.org/iso/catalogue_detail.htm?csnumber=39064. 2006, ISO/IEC.

[8] Newcomb, P., Architecture-Driven Modernization (ADM), in Proceedings of the 12th Working Conference on Reverse Engineering. 2005, IEEE Computer Society.

[9] OMG. ADM Task Force by OMG. 2007 9/06/2009 [cited 2008 15/06/2009]; Available from: http://www.omg.org/.

[10] OMG, QVT. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. http://www.omg.org/spec/QVT/1.0/PDF. 2008, OMG.

[11] Pérez-Castillo, R. MXML to KDM Transformation implemented in QVT Relations. 2011 29/03/2011 [cited 2011 29/03/2011]; Available from: http://alarcos.esi.uclm.es/per/rpdelcastillo/modeltransformations/MXML2KDM.htm.

[12] Pérez-Castillo, R., I. García Rodríguez de Guzmán, and M. Piattini, Architecture-Driven Modernization, in Modern Software Engineering Concepts and Practices: Advanced Approaches, A.H. Dogru and V. Bier, Editors. 2011, IGI Global. p. 75-103.

[13] Pérez-Castillo, R., I. García Rodríguez de Guzmán, and M. Piattini, Knowledge Discovery Metamodel - ISO/IEC 19506: a Standard to Modernize Legacy Systems. Computer Standards & Interfaces Journal, 2011: p. In Press.

[14] Pérez-Castillo, R., B. Weber, I. García Rodríguez de Guzmán, and M. Piattini, Process Mining through Dynamic Analysis for Modernizing Legacy Systems. IET Software Journal, 2011. 5(3): p. 304–319.

[15] promtools.org. ProM tool - Example Logs Files. 2009 [cited 2011 01/04/2011]; Available from: http://www.promtools.org/prom5/downloads/Process%20Log%20examples.zip.

[16] Runeson, P. and M. Höst, Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Eng., 2009. 14(2): p. 131-164.

[17] Sneed, H.M., Estimating the Costs of a Reengineering Project. Proceedings of the 12th Working Conference on Reverse Engineering. 2005: IEEE C. S. p. 111 - 119.

[18] Van der Aalst, W.M.P., Process-Aware Information Systems: Lessons to Be Learned from Process Mining, in Transactions on Petri Nets and Other Models of Concurrency II, J. Kurt and M.A. Wil, Editors. 2009, Springer-Verlag. p. 1-26.

[19] Van der Aalst, W.M.P., B.F. Van Dongenm, C. Günther, A. Rozinat, H.M.W. Verbeek, and A.J.M.M. Weijters, ProM : the process mining toolkit, in 7th International Conference on Business Process Management. 2009, Springer. p. 1-4.

[20] Weske, M., Business Process Management: Concepts, Languages, Architectures. 2007. Springer. p. 368.