Recompiling Learning Processes from Event Logs

Juan C. Vidal[a,*], Borja Vázquez-Barreiros[a], Manuel Lama[a], Manuel Mucientes[a]

[a] *Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)*
Universidade de Santiago de Compostela, Santiago de Compostela, SPAIN

Abstract

In this paper a novel approach to reuse units of learning (UoLs) —such as courses, seminars, workshops, and so on— is presented. Virtual learning environments (VLEs) do not usually provide the tools to export in a standardized format the designed UoLs, making thus more challenging their reuse in a different platform. Taking into account that many of these VLEs are legacy or proprietary systems, the implementation of a specific software is usually out of place. However, these systems have in common that they record the events of students and teachers during the learning process. The approach presented in this paper makes use of these logs *(i)* to extract the learning flow structure using process mining, and *(ii)* to obtain the underlying rules that control the adaptive learning of students by means of decision tree learning. Finally, *(iii)* the process structure and the adaptive rules are recompiled in IMS Learning Design (IMS LD) —the *de facto* educational modelling language standard. The three steps of our approach have been validated with UoLs from different domains.

Keywords: Learning flows discovery, process mining, adaptive rules mining, IMS Learning Design.

1. Introduction

While designing a course, there are two main concerns that worsen the realization of an educational scenario: *(i)* how to model a practical pedagogical scenario to achieve the educational objectives, and *(ii)* how to reuse this scenario in another context than the original. Teachers do not only define the learning content to be consumed by the learners, but they also include the different educational objectives, the order in which the learning activities must be undertaken to achieve these objectives, the evaluation methods, etc. Hence, to reuse and better validate an educational scenario, it should be explicitly written. Although these learning designs, i.e., the descriptions of the educational process, are usually portrayed with documents that use natural language, they can be formally described through Educational Modelling Languages (EMLs). Moreover, when interacting with a virtual learning environment (VLE), learners also perform additional activities than the specifically defined by the teachers, such as interacting in the forum, checking the bibliography, etc. This information should be also highlighted to enable teachers to improve the *learning flow*,

*Corresponding author
Email addresses: juan.vidal@usc.es (Juan C. Vidal), borja.vazquez@usc.es (Borja Vázquez-Barreiros), manuel.lama@usc.es (Manuel Lama), manuel.mucientes@usc.es (Manuel Mucientes)

i.e., the *real* workflow of learning activities, as well as the *evaluation process* [33]. Therefore, the defined educational scenario is more complex than the learning design explicitly documented by teachers.

In the last decade a great effort has been made for developing EMLs. The main idea underlying these languages is to describe, from a pedagogical point of view, the learning process of the course, i.e, the *sequence of steps* the learners should undertake to achieve the educational objectives of the course, by using the available educational resources and services. Regarding the wide variety of specifications for representing learning designs, one standard *de facto* has jumped into e-learning panorama: the IMS Learning Design (IMS LD) specification [8]. IMS LD enables the formal description of learning processes for a wide range of pedagogical contexts in a VLE. Although there is some controversy about whether IMS LD is too complex to be understood by teachers from a practical point of view [11] — especially with the levels B and C—, most of authors highlight this complexity as a barrier for adopting IMS LD [25].

To deal with this issue, a number of user-friendly authoring tools have appeared [19, 14, 16, 9, 24, 5], but even with these tools, authoring process of IMS LD units of learning[1] (UoLs) is not easy for teachers when these UoLs are complex or require to use advanced features of this standard. The *automatic* reconstruction of UoLs could relieve this issue [25], promoting the use of IMS LD by teachers and instructors. Taking as starting point the event log files, which stores all the events generated by the learners, it is possible to mine the *real behavior* undertaken by the students during the UoL, i.e, what the learners really did, and the rules that constraint the behavior of the model. Then, by combining these two models, it is possible to reconstruct the UoL to a specific target language. Therefore, this process facilitates the reuse of defined UoLs no matter the VLE that has been used, as the techniques used for both mining the variable values and the formal model are totally independent of the domain. Therefore, teachers can design their courses within their VLE, avoiding the need to use an authoring tool with a specific EML notation, and still be possible to reconstruct the UoLs from the scratch to a target language — such as IMS LD.

In this paper, we present an approach to *automatically* reconstruct the IMS LD representation of an UoL from the events generated by the learners in the VLE. This objective is achieved in three different steps. Firstly, the learning flow of the UoL is automatically extracted from the logged sequences through a *process discovery algorithm*. Then an algorithm based on the knowledge about the IMS LD control structure is applied to determine which IMS LD components should be created. Finally the adaptive rules of the UoL are automatically extracted form the event logs —more specifically, from the variable values of the logs— by a decision tree learning algorithm, and integrated into the IMS LD structure. The contributions of this proposal are: *(i)* a new framework to facilitate the reuse of UoLs between different VLEs; *(ii)* the automatic discovery of learning processes from event logs and its recompilation to IMS LD; and *(iii)* the automatic identification of the adaptive rules from event logs.

Notice that IMS LD has a high expressiveness to allow the definition and orchestration of complex activity flows in a multi-role setting, but at the expense of complexity. In fact, current IMS LD research seems to accept the

[1] An UoL represents a variety of prescribed activities, assessments and services provided by teachers, in a course or lesson, which is the result of the learning design.

assumption that specification's conceptual complexity difficult the authoring process [11]. Taking this into account, another objective of this paper is to reduce this barrier and facilitate the adoption of UoLs specified in IMS LD by instructors. Specifically, the proposed semi-automatic approach hides the complexity of the EML language, so instructors only have to decide which one of the recompiled processes fits better with the learning objectives of the UoL, in terms of structure and adaptive criteria. Henceforth, the main research question addressed in this paper is the automatic reconstruction of UoLs from scratch, as the state of the art heavily relies in the participation and feedback from all appropriate personnel and users during the whole process, hindering the reuse of UoLs in different platforms.

The remainder of this paper is structured as follows. Section 2 briefly introduces the main features of the IMS LD specification. Section 3 describes the different approaches that have already been proposed and that motivated our approach. Then, Section 4 presents the framework that supports the mining of log files and the reconstruction of IMS LD. Sections 5 and 6 detail, respectively, how the learning flow —through a process discovery algorithm— and the adaptive rules —through a decision tree algorithm— are mined from the logs. Then, Section 7 details the transformation from these two models to the actual target language (IMS LD). Section 8 shows the results and, finally Section 9 points out the conclusions and future work.

2. IMS Learning Design

IMS LD specification is a meta-data standard that describes all the elements of the design of a teaching-learning process [8]. This specification is based on: *(i)* a well-founded *conceptual model* that describes the vocabulary and the functional relations between the concepts of the learning design; *(ii)* an *information model* that details in natural language the semantics of every concept and relation introduced in the conceptual model; and *(iii)* a *behavioral model* that specifies the constraints imposed to the software system when a given learning design is executed in run-time. In other words, the behavioral model defines the semantics during the execution phase. Furthermore, IMS LD defines three levels of implementation depending on whether the learning design is adaptive or not:

- *Level A*. This first level contains the main components of a UoL: participants (roles), pedagogical objectives, resources (services and contents), and learning design. This last component is understood as the coordination of the learning activities to be performed by the participants to achieve the pedagogical objectives, i.e., the learning design describes the learning flow –or learning path– to be followed by learners in a UoL. To describe this learning design, the IMS LD specification follows a theater metaphor where there are a number of plays, that can be interpreted as the runscripts for the execution of the UoL and that are *concurrently* executed, being independent of each other. Each one of these plays is composed by a set of acts, which can be understood as a module or chapter in a course. Acts are performed in *sequence* and define the activities that participants must do. This model also allows the assignation of roles to the participants and partitioning the activities of an act according to those roles. In this case, each one of the partitions can run *in parallel*. Finally, activities can be simple or complex, the latter may consist of a *sequence* or *selection* of activities (simple or complex).

- *Level B*. This level adds *properties* and *conditions* to level A. It also adds monitoring services and global elements which allow users to create more complex structures. The properties store information about people (preferences, outcomes, roles, etc.), personal information, or even about the learning design itself. Level B also establishes *(i)* the visibility of the elements of the learning flow; *(ii)* if properties are transient or should persist across multiple sessions; and *(iii)* the set of operators and expressions that may transform the value of properties and the visibility of elements. For instance, adaptation is usually based on the visibility of the activities of the learning flow, since IMS LD does not have control structures such an *if-then-else*. Therefore, the adaptation rules use properties, such as a test score, an answer to a specific exercise, and so on, to decide the learning path of the student through the visibility of the activities.

- *Level C*. The last level incorporates notifications to level B. Notifications fire automatically in response to events triggered in the learning process. For example, if a student submits a job, an email to report the event could be automatically sent to the teacher.

Taking this into account, the objective of this paper can be defined more precisely as recompiling the structure of the learning process defined at level *A* and the properties and adaptation rules at level *B* from event log files.

3. State of the Art

We have focused our analysis of the state of the art in the topics and fields that motivated our approach: *(i)* IMS LD authoring tools; *(ii)* the reconstruction of IMS LD; and *(iii)* the applications of process mining in education.

For the last years, a number of IMS LD authoring tools have been devloped. These solutions allow a better analysis of the related educational design approaches by trying to relieve the complexity of this standard to teachers and instructors. ASK-LDT [19] provides a graphical interface that allows to hide the complexity of the IMS LD control structure and adaptive components. The authors define an abstract high-level architecture for designing pedagogical scenarios that can be reused in different virtual learning environments. In [15], the authors offer visual templates or patterns for creating learning designs based on pedagogical strategies such as collaborative learning or project based learning. Within this idea, they present Collage, a specialised high-level collaborative learning editor that guides teachers to create their own collaborative learning design from the existing patterns. However, this tool is not intended to edit templates, so it is restricted to the predefined set. Other tools such as ReCourse [14] or Prolix GLM [16] made a great effort by providing the user-friendly graphical notation, however they do not have wide support for level B, i.e., they have difficulties providing a visual notation for level B properties and conditions [12]. Other tools, such as LAMS [9] and MOT+ [27], although they are not based on IMS LD, can handle this standard by making a translation to their internal EML representation. Nonetheless they can lack of flexibility —being difficult to interpret by non-experts— or face difficult importation/exportation issues in order to interoperate with IMS LD. In summary, although these approaches try to hide the complexity of IMS LD by using a more user-friendly notation, they share the same

drawback: real users —teachers and instructors— do not have the technical skills needed for a practical use of these tools, as they still require to know about IMS LD to use the advanced features of this standard. In fact, this drawback works as a barrier for the practical use of IMS LD, as teachers are forced to transform their educational scenario into machine-oriented notation [32].

The automatic reconstruction of UoLs is a novel approach to hide the complexity of the authoring process of IMS LD. In [25], the authors present an approach that makes a conceptual distinction between exchange EMLs and authoring EMLs. In this approach, the authors provide graphical notations —authoring EMLs— to the teachers, which are more user-friendly than the raw XML format. Then, these notations are translated, via an exportation process. With this classification, they focus on the reengineering of IMS LD UoLs based on a visual language, which hides the complexity of the model. On the other hand, this approach also requires a close collaboration between developers and teachers to simplify the gap between the technical and pedagogical point of view of the UoL. In [6], and later in [5], the authors present a four-step approach for process reenginering in higher education. However, the reengineering process is not fully automatic, as it requires the participation and feedback from all appropriate personnel and users. In summary, very valuable results have been achieved in this field, however, the main drawback of these approaches is that the UoL is not automatically reconstructed from the scratch and needs the supervision of teachers and even developers. From the point of view of mining the control flow of processes, a recent approach [22] tries to mine the adaptive rules of the log by using decision trees. In this approach, the authors define a framework for deriving and correlating process characteristics.

Process mining has emerged as a way to analyze the behavior of an organization based on event logs. Specifically, process mining focus is on concurrent processes, trying to discover the underlying control flow of the behavior recorded in a event log: *sequences*, *parallelisms*, *loops*, etc. On the other hand, a second important distinction concerns the unit of analysis, which can be variables or events. Most of the approaches found in the literature use events for automatically mining the processes; however, this second dimension depends directly on the information available in the log files. Taking these features into account, and although process mining from log files has been widely applied [10, 23, 1], its application in education is a relatively new topic with some recent studies. In [20], the authors present PETRA, a system to extract new knowledge rules about transitions and learning activities in processes from previous platform executions. However, this tool is not oriented towards process reconstruction and discovery, but on process extension, i.e., it requieres an already defined process model in order to enrich such process. In [17] a process mining approach in adult informal learners is presented. Informal learning situations often exhibit high variability within the learner population, especially when learning experiences and environments offer broad availability. However, this freedom of navigation has sometimes negative effects of learning experiences, particularly when prior domain knowledge or learning skills are weak. The experience is developed in two steps: firstly, a process discovery using hidden Markov models is performed [18], and then a process analysis is done using a standard process mining approach [3]. A similar approach is presented in [29] but for improving on the objectivity of the assessment as well as to provide the learners with prompt feedback. A set of software repositories are preprocessed with the FRASR

tool [30], to produce event logs and then mine them using ProM [13]. The first step of this approach is particularly interesting and differs from previous proposals. The authors do not start the derivation from a log file but from a set of repositories that have different structures. Another process mining approach for providing students feedback is described in [28]. In this case, the objective is to discover a process from the behavior of the students during the course of a MCQ test. They analyzed the navigation flow of each student when answering the questions, to determine what kind of feedback is more preferable and more effective for the students. Authors use the α-algorithm [4] for the process discovery. This three approaches are very similar. In fact they almost use the same process mining techniques. The main drawback is that the process discovery techniques that they use do not always guarantee feasible results.

In summary, although a wide variety of approaches have tried to provide a IMS LD user-friendly notation by means of graphical tools, its complex specification, particularly the levels B and C, entails a barrier for adopting this standard by teachers and instructors. The automatic reconstruction of IMS LD UoLs could relieve this issue, but it is necessary to deep in the automation of this process from the scratch, avoiding the supervision by teachers and developers during the whole process, and thereby facilitating the reuse of UoLs in different platforms.

4. Framework

Figure 1 depicts the conceptual framework for reconstructing IMS LD UoLs from the events generated by learners. It shows all the main parts involved in the reconstruction of the IMS LD UoLs. The first component of this framework is the *educational world*. Teachers and learners are the typical participants in any learning activity. On the one hand, *teachers* design the learning flows based on some educational methodology, and support the learning activities of the course. On the other hand, *learners* are the core of the educational world since they undertake the learning flow activities by using the resources and services available in the virtual learning environment. The rest of the components are: the *virtual learning environment*, the *event log system*, the *learning flow discovery*, the *the learning flow hierarchization*, and the *adaption rules*.

- *Virtual Learning Environment*. From an educational point of view, virtual learning environments (VLEs) provide the means to carry out the learning activities planned for an UoL, allowing learners to access to the learning contents and executing the services required to facilitate those activities such as interacting with other learners or looking for information in libraries. Furthermore, VLEs detect and register all the relevant events generated by learners when undertake the learning activities. These events are stored in an event log database that contains data about the activities execution, including who participates in an activity, when it has been performed, the properties values of the UoL such as a mark of a test, and so forth.

- *Event Log System*. When a learner undertakes a learning activity, such as answering a test, uploading an exercise or even asking a question in the forum, the VLE stores in a database the information generated as result of performing this learning activity. However, in order to be able to extract the needed information for

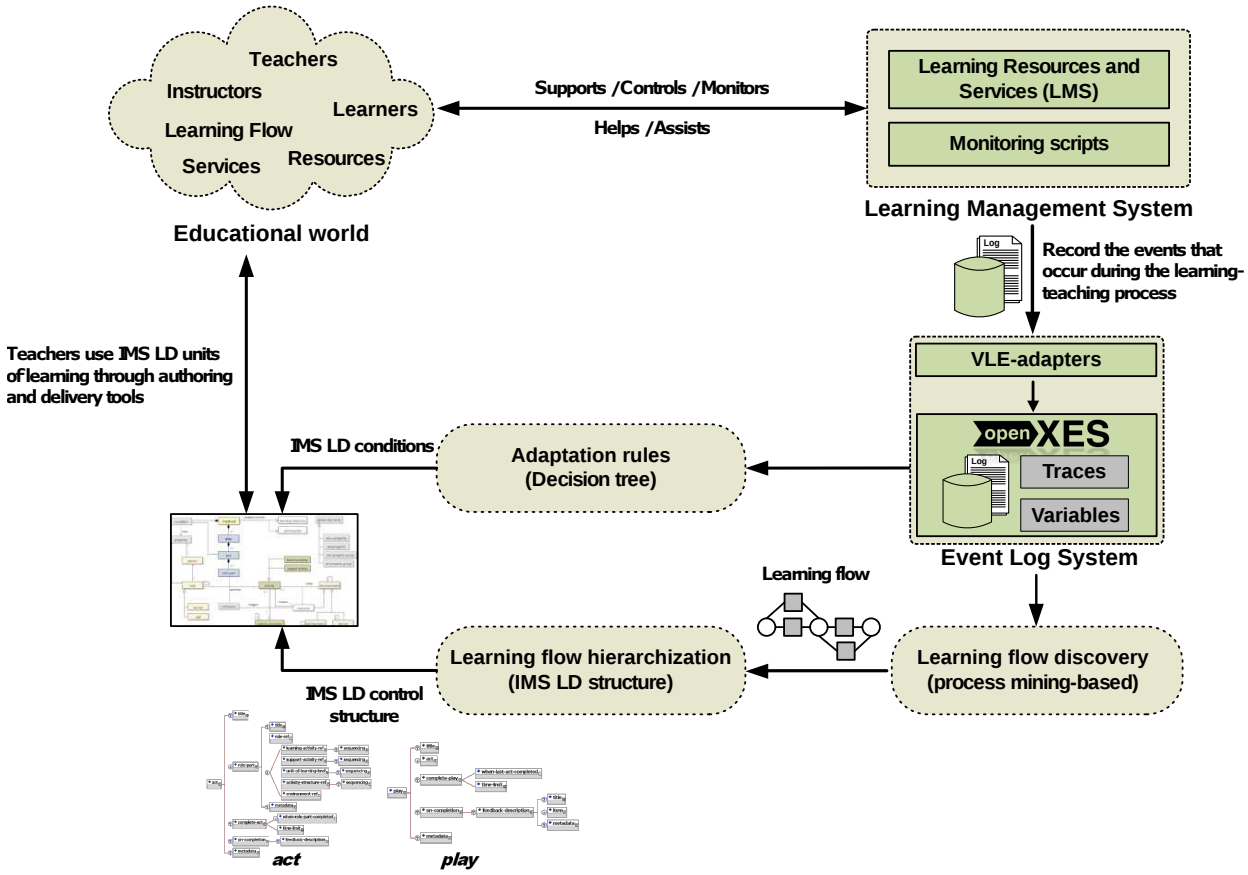


Figure 1: Framework for IMS LD UoLs reconstruction.

reconstructing the IMS LD, it is necessary to translate this information into an input format for both the process mining and the decision tree algorithm. As Figure 2 depicts, this translation is carried out by VLE-specific *adapters* that generate an event log register in a standard format for process mining, so-called *eXtensible Event Stream*[2] (XES) format [37]. XES is a well-known logging standard for process mining, and we have adapted it with user-defined extensions in order to include in the same file all the outstanding information to both obtain the learning flow —by storing the sequence of steps undertaken by each learner— and the adaptive rules —by storing the information generated as result of performing the learning activities. Figure 2 shows an example of the framework used to translate the event log from any VLE to an event log in XES format[3] through specific VLE-*adapters*. These adapters and the use of XES standard provide the independence of reconstructing the IMS LD with the particular VLE in which the learning-teaching process takes place, enabling furthermore to generalize the proposed architecture to any virtual environment. The only restriction required for the VLE is that it must register the user activity —along with the variable modifications— that takes place in the learning

[2]www.xes-standard.org

[3]Note that we have omitted some of the standard metadata of XES in the event log shown in Figure 2 to make the image more readable.

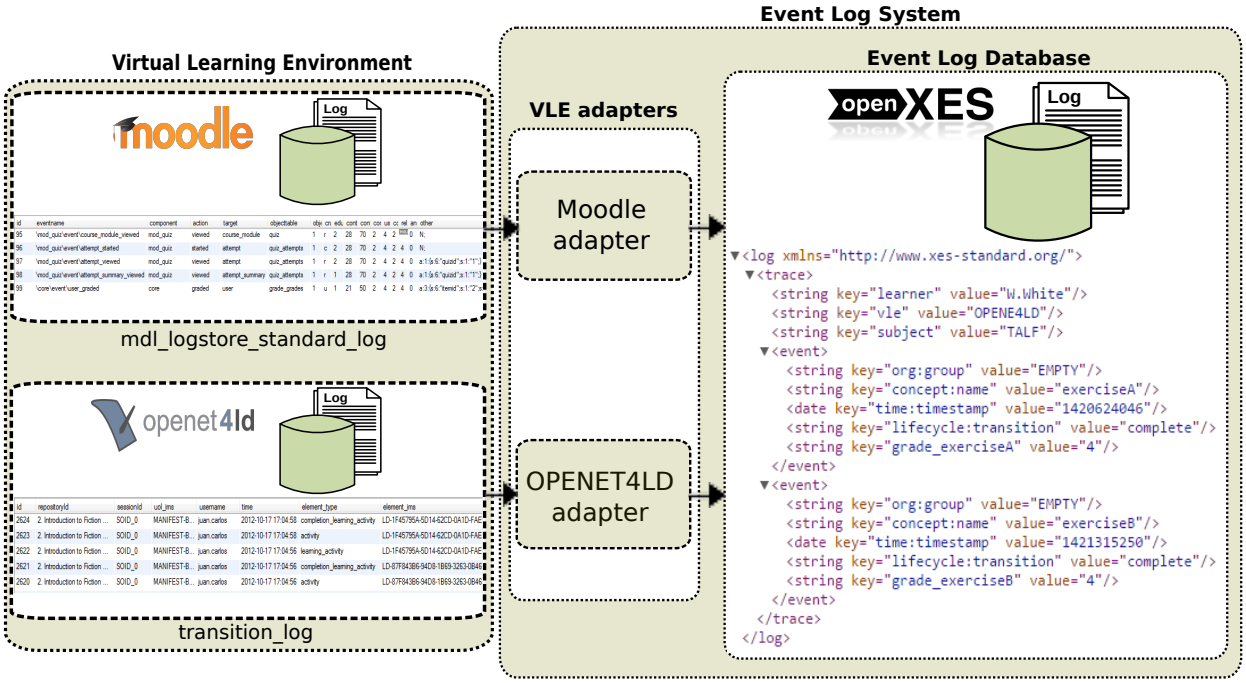


Figure 2: Event log infrastructure. Each VLE accesses the event log system through a specific adapter to transform its log format to XES, the standard format for process mining.

context.

- *Learning Flow Discovery*. This component implements the algorithm whose aim is to discover the workflow of learning activities, i.e., the learning flow, that learners undertake during a UoL. Note that this algorithm must guarantee that *all the learning paths* followed by learners are represented in the discovered learning flow. Furthermore, this discovered learning flow should be as simple as possible in order to facilitate the hierarchization of the learning flow into the IMS LD control structure. This component is explained in detail in Section 5.

- *Learning Flow Hierarchization*. Once the learning flow has been extracted from the event logs, an algorithm will translate this learning flow to the control structure defined by the IMS LD specification. This algorithm starts by detecting sequences and selections of learning activities and then uses the knowledge about IMS LD to create activity structures, acts , and plays. The learning flow hierarchization is explained in detail in Section 7.

- *Adaptation Rules*. The IMS LD specification has an extensive set of adaptation conditions, but this framework is focused on extracting the conditions related to the selection of learning activities — show and hide mechanism — based on the changes in the properties values of the UoL. A decision tree technique was implemented to automatically obtain these conditions, since it is an effective approach to deal with this kind of problems. This process is explained in Section 6.

It is important to emphasize that, although we are depicting this framework for IMS LD UoLs reconstruction, in

fact, the framework described can be applied to reconstruct any UoL to any particular target EML. Both the process mining step and the adaptive rules mining from variable values, are totally independent from the target language. In fact, process mining retrieves graph-based structures independent from IMS LD. On the one hand, graphs are based on Petri nets, which is the most used formalism for modelling processes. On the other hand, a decision tree algorithm is used to get the adaptive rules that will guide the students along the UoL. The learned knowledge is represented through binary trees and, thus, not tied to the specificity of IMS LD rules.

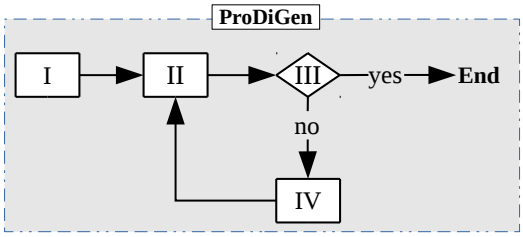
The last step of this framework, i.e, the combination of this two models into the target language, is the only dependency between the reenginering process and the target EML. Nevertheless, the algebra that we are using —explained in Section 7— in the reconstruction of the IMS LD is not so different from any particular EMLs —*if-then-else*, *AND*, *OR*, *sums*, etc—. Hence, it is possible to modify the knowledge applied in the last step of the reeinginering process in order to reconstruct an UoL *from any particular VLE to any particular EML.*

5. Mining the learning flow from event logs

The goal of process mining is to automatically obtain a process model that specifies the relations between activities from *concurrent* processes. Therefore, the aim of the process discovery algorithm in the reconstruction of the IMS LD is to identify the workflow that represents the learning flow followed by the learners during the UoL [7]. To achieve this objective, the process discovery algorithm only needs to consume the log in XES format —as this is the standard format for process mining— and it will retrieve a learning flow representing the behavior recorded in the event log. From the perspective of process mining [35], the quality of a learning flow is measured taking into account the following criteria:

- *Fitness replay*, which indicates how much of the behavior observed in the event log can be reproduced by the discovered learning path. A discovered learning path is considered complete when it can reproduce all the events contained in the log database. In order to guarantee a feasible and correct reconstruction of IMS LD UoLs, all the activities undertaken by the learners have to be included in the mined learning flow. Additionally, from the educational point of view, in order to guarantee feasible and correct evaluations of the learning paths, teachers need to access to all the activities performed by learners. Therefore, the fitness replay of discovered learning paths is a hard requirement.

- *Precision*, which measures if the discovered learning path allows an additional behavior that is not represented in the log, i.e, behaviour never undertaken by the students. Thus, a discovered learning path is considered as precise when it cannot reproduce events that are not available in the log database. From the point of view of the learners evaluation, this kind of learning paths are desirable, but it is not a requirement as hard as the fitness replay: the additional learning paths are not needed for the reconstruction of the IMS LD control structure, since they did not happen. Our focus is on the exact behavior of the learners, i.e, *what they really did*, not in the prediction of their behavior; therefore we want to avoid overly-general models.



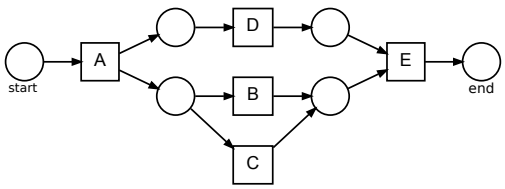
Step	Description
I	Build the initial population
II	Evaluate each solution
III	Stopping conditions fulfilled?
IV	Generate the new population

Figure 3: Main steps of ProDiGen.

- *Simplicity*, which refers to discovered learning paths with the minimal structure that reflects the behavior contained in the log database. A desirable requirement for process discovery is to obtain simple learning flows, since the simpler is the discovered learning flow, the easier is to reconstruct the IMS LD control structure.

In general all process discovery algorithms make assumptions regarding the event log and, hence, the emphasis on these three different quality dimensions. For example, Heuristics Miner [39] usually retrieves models with high levels of precision, but lacking fitness replay and/or simplicity. Another example is the ILP miner [40] that guarantees a perfect replay fitness but resulting in very complex models. Inductive Miner [21], on the other hand, can guarantee a perfect replay fitness and high levels of simplicity, but the precision is, overall, low. Overall, a large amount of work has been done in this specific area by addressing different algorithms from different points of view. However, in this paper, and considering the three previous criteria and their importance when mining a learning process, we selected ProDiGen [36] as process mining algorithm.

ProDiGen is a genetic algorithm for process discovery that focuses its search towards solutions that replay all the behavior as possible, with *high levels* of precision and simplicity. In order to better understand how this algorithm works, Figure 3 shows a simplification of its mains steps: *(i)* the initialization of the population; *(ii)* the evaluation of the individuals; *(iii)* the stopping criteria; and iv) the generation of the new population. The evolutionary cycle involves the II, III and IV steps, where the population evolves based on a fitness function. Finally, the algorithm stops whether it reaches a maximum number of generations, or it gets stuck on a local minimal solution for a certain number of times.



(a) *Example of a Petri net.*

Task	I(Task)	O(Task)
A	{}	{{D},{C B}}
B	{{A}}	{{E}}
C	{{A}}	{{E}}
E	{{D},{B,C}}	{}
D	{{A}}	{{E}}

(b) *Causal matrix of the Petri net.*

Figure 4: Mapping of a Petri net into a causal matrix. This Petri net represents the learning flow of an UoL about polymorphism. The name of the activities are: Read about polymorphism (A), Exercise 1 (B), Exercise 2 (C), Answer test (D) and Exam (E).

5.1. ProDiGen

The first step in any evolutionary algorithm is the initialization of the population. In this phase, a population is created with a group of individuals where each individual is a potential solution, e.g, a learning flow. In ProDiGen, each individual of the population codifies a learning path using a causal matrix representation, which can be easily translated into a Petri net [26]. In terms of causality, both the Petri net and the causal matrix represent the same behavior —which learning activities enable the execution of other learning activities. Figure 4 shows an example of the mapping between a Petri net (Fig. 4a) and its respective causal matrix (Fig. 4b). This causal matrix has a row for each learning activity t in the log, and two columns corresponding to the inputs, $I(t)$, and outputs, $O(t)$, of the activity. The input and output sets are composed of several subsets, modelling the following relations:

- In the same subset:
  - Tasks in the same subset of the conditional function $O(t)$ have an OR-split relation.
  - Tasks in the same subset of the conditional function $I(t)$ have an OR-join relation.
- Between different subsets:
  - Tasks in different subsets of the conditional function $O(t)$ have an AND-split relation.
  - Tasks in different subsets of the conditional function $I(t)$ have an AND-join relation.

Once the initial population is created through heuristics based on the local information of the log, the individuals —causal matrices— of the population are modified through crossover and mutation operations to create new potential solutions, i.e, learning flows. These operators add and/or remove[4] causal dependencies of the individual by modifying the input and output conditional functions of the tasks. Then, these new individuals are evaluated based on a fitness function.

In ProDiGen, the quality of an individual is measured taking into account fitness replay, precision, and simplicity, as ordering criteria, respectively. Hence, when contrasting two possible solutions, fitness replay is the primary ordering criteria. When two solutions have the same fitness replay, precision is used to decide the best solution. Finally, when two solutions have the same fitness replay and precision, simplicity is the decisive criterion.

Thus, when two solutions have to be compared, the individual with highest fitness replay will be the benefited one. If the values for the fitness replay are equal for both solutions, the second parameter to be checked is the precision, and, if the same as before occurs, the last one to be compared is the simplicity. Note that if we change the hierarchical order of the fitness measure, the algorithm may find a different solution, as fitness replay, precision and simplicity are three opposed objectives [36].

[4]For example, the mutation operation can add a new subset into the output function of a task t, resulting in an AND-split.

(3, 'UOLID_201', 'GeoQuiz3', 'root', '2013-10-14 01:44:51', '13193839-3ddf-41e2-a4b8-7af8dc13d059'),

(4, 'UOLID_201', 'GeoQuiz3', 'root', '2013-10-14 01:44:52', 'e4901d0f-8232-4cce-a166-20e29133d279'),

(5, 'UOLID_201', 'GeoQuiz3', 'root', '2013-10-14 01:44:52', 'e4901d0f-8232-4cce-a166-20e29133d279'),

(6, 'UOLID_201', 'GeoQuiz3', 'd9', '2013-10-14 01:44:52', 'e4901d0f-8232-4cce-a166-20e29133d279'),

...

(289, 'OPENET_RMI_SOID_0', 'UOLID_201', 'setproperty', '2013-10-14 01:59:30', 'user__2', 'd2', 'Student', 'locpers_property__12', 'Answer1', 'locpers_property', 'string', 'Venezuela'),

(290, 'OPENET_RMI_SOID_0', 'UOLID_201', 'change_property_value__0', '2013-10-14 01:59:30', 'user__2', 'd2', 'Student', 'locpers_property__1', 'Value1', 'locpers_property', 'integer', '2'),

(291, 'OPENET_RMI_SOID_0', 'UOLID_201', 'change_visibility', '2013-10-14 01:59:30', 'user__2', 'd2', 'Student', 'learning_activity__5', 'flow3', 'locpers_property', 'boolean', 'false'),

(292, 'OPENET_RMI_SOID_0', 'UOLID_201', 'setproperty', '2013-10-14 01:59:33', 'user__2', 'd2', 'Student', 'locpers_property__13', 'Answer2', 'locpers_property', 'string', 'Siria'), ...

Figure 5: Example of a text-based log file

6. Mining adaptive rules from event logs

Using log files of VLEs can help to determine who has been active in the course, what they did, and when they did it. In this section we use these data to obtain the adaptive rules of the UoLs that determine the learning flow and the contents and services presented to students. Unfortunately, the log files provided by VLEs do not follow a standard and therefore a generic solution cannot be formulated for such purpose. In fact, log files are seldom used mainly because it is difficult to interpret and exploit them. In most of the cases, the data aggregated are incomplete or even not logged.

Taking this into account, in this section we describe our approach based on the extract of the log file represented in Figure 5. This log was recorded in OPENET4LD [38], and exemplifies the typical elements saved by VLEs. Of course, the records formats or even how they are stored may vary —e.g., Moodle saves this information in tables of a relational database—, but usually VLEs present very similar information. This particular example has two parts. On the one hand, the first four records represent the execution of activities and contain information about the UoL id, its name, the user that performed the activity, the execution time, and the specific name of the activity, among other information. On the other hand, the last four records represent events on the properties used by the UoL. Specifically, each one of these records contains information about the UoL id, the operator that favor the change, the type of the property, and the new assigned value.

6.1. Identification of variables and activities

The identification of variables and activities is highly dependent on the type of events recorded in the log files. Since each VLE may record the events in a different format, the syntactic patterns used to identify these events are usually different. For example, the identified variables are highlighted in Figure 5. In this case, a simple regular expression with the keywords ”setproperty” and ”change_visibility” were used to identify the variables, but a different pattern should be used to identify these same variables in an XML-based log file. However, the main issue in the identification of variables is that they are not always recorded in the log files. When this situation happens, the mechanism for learning the adaptation rules presented in this section cannot be applied.

6.2. Determining the variable values for each activity

In the ideal situation, each time an event is produced, *(i)* the state of the variables is saved in a log file. This means that, e.g., at the end of an activity the values of the variables of a UoL are stored in the corresponding log file. Moreover, *(ii)* in this context a value change is also considered an event and so is also recorded. However, reality is different and usually both mechanisms are not supported at the same time —e.g., in the log extract of Figure 5 only variable changes are included.

The variables values are subsequently associated to an activity, so we can determine the state of the properties before and after the activity is performed. Therefore, each time a variable value changes we must determine when and by who it was modified. In this procedure, the time of the event is crucial since it will determine the initial value of variables in the next activity.

6.3. Learning rules with a decision tree

IMS LD rules use the following grammar to define the learning flow adaptation:

```
rule ::= IF <expression> THEN <action>
       | IF <expression> THEN <action> ELSE <action>
       | IF <expression> THEN <action> ELSE <rule>
```

In addition, IMS LD declares three types of actions:

```
action ::= SHOW <activity_id>
       | HIDE <activity_id>
       | <property_id> = <value>
```

The first and second actions are used to make visible or invisible a learning/support activity, activity structure, play, item, or environment. It’s the main adaptation mechanism of IMS LD used to hide or show a part of the UoL based on some expression value. The last type of expression is a simple assignment.

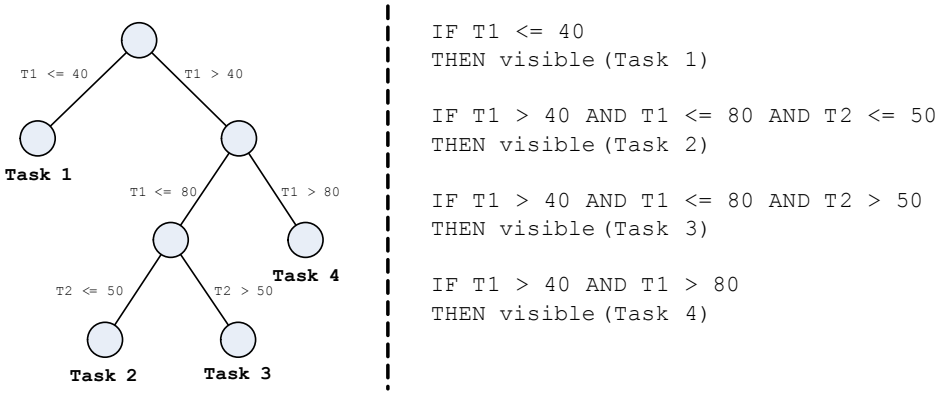


Figure 6: Transformation of a decision tree to a DNF rule base

IMS LD grammar for expressions include logical operators, *and*, *or*, and *not*, some comparative operators, $\leq, <, =, \neq . >$, and $\geq$, and some multiplicative and additive operators. Moreover, it also defines operators to check specifics of UoLs. For instance, to verify if a play, act, or activity has already finished, or, e.g., the specific role of users.

In this paper, the identification of the adaptive rules is performed by means of J48 decision tree algorithm [31]. We selected this type of algorithm because of its simplicity, performance, and especially because adaptive rules can be transformed to a decision tree. As previously mentioned, IMS LD adaptive rules have an *if-then-else* structure very similar to the right part of Figure 6, which can be deduced from the graph structure returned by a decision tree algorithm. A decision tree is a graph-like structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test and each leaf node represents the class label —or decision taken after computing all attributes. A path from the root to the leaf represents classification rules, and in our case, an adaptive rule.

Transforming a decision tree to an adaptive rule is straightforward since these trees can easily be modelled as DNF (Disjunctive Normal Form) rules. For instance, the right part of Figure 6 shows the corresponding rules for this tree. Specifically, each branch is converted into a rule, where:

- The condition is set as the conjunction of the arc tests of a branch.
- The action of the rule is the leaf node of the branch —class/decision.
- The disjunction of all the rules has the semantics of the decision tree.

Notice that the rules extracted by J48 support most of the grammar of IMS LD —logical and comparative operators—, although this is not the main objective here. In fact, we selected decision trees because they provide a simple but also generic grammar, that should be compatible with most of the adaptation mechanisms used in legacy systems. Therefore, it does not make sense to learn, e.g., IMS LD specific operators. There are however some limitations since we do not cover complex conditions that combine mathematical operators.

Tree complexity has its effect on the accuracy and is usually determined by the total number of nodes, total number of leaves, depth of tree, and number of attributes used in the tree construction. The number of variables is therefore a crucial factor since too many of them may reduce the accuracy of the tree. Moreover, the size of the data required to learn increases with the number of variables. However, adaptive rules are usually not based on many variables. Thus, our approach limits the variables considered during the learning of the decision tree, and specifically, only variables for which the value has changed in the execution of the activity are included.

7. IMS LD reengineering

IMS LD is a well-known EML for adaptive learning which is specified in three different levels of implementation —levels A, B, and C—, which determine the learning flow, the changes in the environment, and the notifications, respectively. In this section we describe an algorithm that compiles IMS LD levels A and B from the information retrieved by the data mining algorithms described in the former sections. On a first step, the proposed algorithm transforms the flat process structure retrieved by the process mining into an IMS LD-based structure. On the second step, the identified adaptation rules are associated to the learning flow.

7.1. Pairing the causal matrix with the IMS LD specification

IMS LD learning flow is described as a theatre metaphor where there is a number of plays that are *concurrently* performed, being independent of each other. Each of these plays is composed of a set of acts, which represent, for instance, the modules or chapters of a course. Acts are performed in *sequence* and define the activities that participants must do. This model also allows the assignation of roles to the participants and partitioning the activities of an act according to that roles. In this case, each one of the partitions can run *in parallel*. Finally, activities can be simple or complex, the latter may consist of a *sequence* or *selection* of activities (simple or complex).

Taking this structure into account, the reengineering process would consist in identifying the different IMS LD elements from the causal matrix (Petri net) returned by the process mining algorithm. However, this Petri net is a flatten process while the IMS LD is a tree-based structure in which each layer is composed by a different type of elements, i.e., first plays, then acts, role-parts, and finally activities. Bearing in mind that the Petri net only identifies the atomic activities, we decided to structure the search space as a tree in which:

- Each node of the tree represents how learning activities are grouped. Suppose an ordered list of the n activities that must be performed. This list can be divided in $n - 1$ parts —one for each consecutive activity. Taking this into account, each node is identified by $n - 1$ digits, where a 1 in the position i indicates that the activities in the positions $i - 1$ and i are in different groups.

- Each layer of the tree corresponds to a layer of IMS LD. Specifically, the first layer represents plays, the second acts, the third role-parts, and the remaining layers activities, simple or complex.

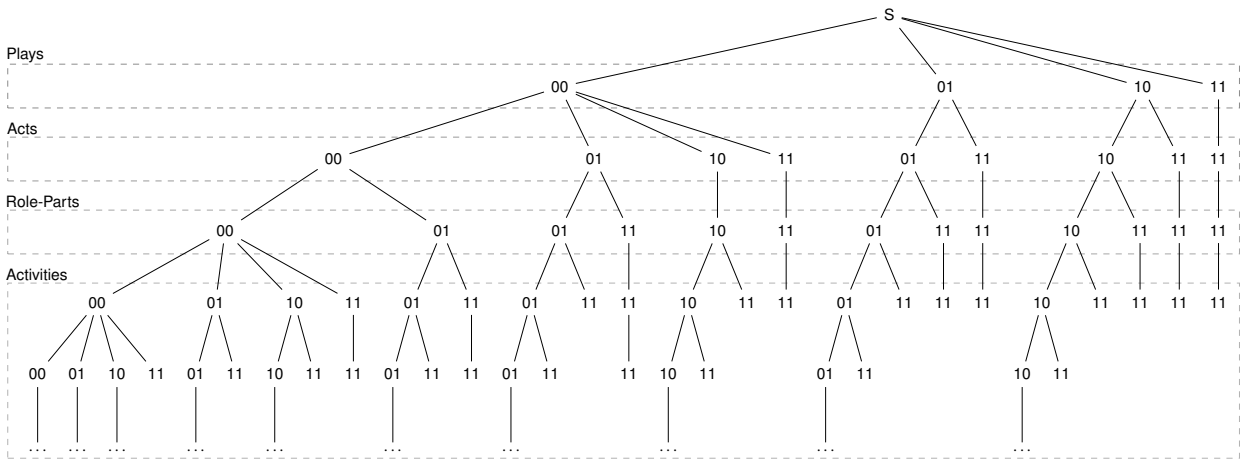


Figure 7: Example of the search space for a UoL composed of three activities

- An edge between a father node and a child node indicates that this child is grouped according to the configuration defined in the father node. Thus, the child node may add additional groups but they must respect the groups defined in the father node.

Figure 7 depicts the search space for a UoL composed of just tree simple activities, namely A, B, and C, ordered according to their position in the causal matrix. Let use the notation $[A\ B\ C]$ to facilitate the definition of the list and use the symbol | to indicate a partition. Taking into account our previous definition of the search space, in this example, nodes can be identified by two digits where: 00 indicates that the three activities are in the same group —$[A\ B\ C]$; 01 that there are two groups, a first one with the activities A and B, and a second one with C —$[A\ B|C]$; 10 also represents a two groups be in this case the activities B and C compose the second group —$[A|B\ C]$; and finally, 11 implies that each activity is in a different group —$[A|B|C]$.

Notice that the meaning of the coding is different for each level of the search tree. For instance, a node identified by 10 at the play level indicates that the activity A is in the first play, while the activities B and C are in the second play. Thus, the coding defines both the number of plays and how activities are grouped in the plays. If the node identified by 10 is at the act level, the node still identifies two groups of activities but in this case for the two acts defined in this coding.

In addition, child nodes must preserve the groups already defined in the father node. Let A and B be two nodes, where A is father node of B in the search tree, and $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_2^n$ denote the coding of size n of A and B, respectively. For each $a_i = 1 \in \mathbf{a}$ then $b_i = 1 \in \mathbf{b}$, $i = 1 \dots n$, i.e., each position i equals to 1 in the father node is also equal to 1 in the child node. This definition ensures the hierarchical disjointness of groups, that is, that two activities that are in different groups cannot be joined in the same group in a lower level of the tree. This constraint also reduces the search space since many father/child combinations are not allowed. For instance, a node 01 can only have two children, denoted as 01 and 11, since the father node already defines two groups —$[A\ B|C]$. Suppose that 10 is a valid child

Algorithm 1: explore

Input: n_i is the node to explore, *verify* is a list of functions used to check the structure of the nodes.
Output: from n_p hangs a (sub)tree of possible UoL configurations that can be derived from the causal matrix.

```
1  verify ← function from verifiers used to check the nodes of the level in which (n_i) is situated
2  if verify(n_i) then
3      if level(n_i) < MAX and not solution(n_i) then
4          foreach j in expand(n_i) do
5              n_j ← new node which value is j
6              add n_j as children of n_i
7              explore(n_j, verify)
8  else
9      if n_i has a parent then
10         remove n_i from parent(n_i)
```

node of 01, this would imply that the activities B and C are in two disjoint groups in the father node but in the same group in the child node, which is clearly inconsistent with the hierarchical definition of the IMS LD specification.

Algorithm 1 details the main block of the depth-first search procedure. The algorithm receives two inputs: n_i which contains the node that is being evaluated, and the *verifiers* list which elements are the functions used to check the structural consistency of the node. Specifically, each function uses the causal matrix obtained during the process mining step to check if the activities are correctly grouped, that is, *(i)* if they verify the dependencies of the causal matrix and *(ii)* the structure is compliant with IMS LD. Since the structural requirements of each level of the tree are different, the *verifiers* list contains a specific verifier for plays, acts, role-parts, and activities. For the sake of simplicity, the position i of the *verifiers* list contains the function to check the level $i+1$ of the search tree. Since there are only 4 types of verifiers, nodes which level is greater than 3 will be checked by the last function of the list —i.e., as activities.

In the first line, the algorithm used the level of the node in the tree to select the corresponding verifier, and uses this function to check the node. This is the most complex part of th algorithm and consists in looking for specific structural patterns in the causal matrix. As we will detail in this section, each IMS LD control construct has structural constraints that are synthesized in one of these functions. When n_i is a solution (i.e., if it is a simple activity) or the MAX level limit has been reached, then the algorithm backtracks. Otherwise, the node is expanded (lines 4-7) and a new exploration starts for the children of n_i.

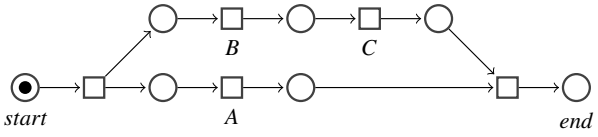
7.2. Consistency of plays

The verifier of the first level of the search tree checks if the plays are grouped in conformance with the IMS LD specification. Specifically, the objective of Algorithm 2 is to verify that the activities contained in each group are in parallel since each group identifies a play. Therefore, in lines 2-11, each activity a_j of the group g_i is compared with a different activity a_l of a different group g_k, until all the activities of the different groups have been compared. In order to check if two activities are in parallel we use the causal matrix M obtained during the process mining step. In this

Algorithm 2: plays

Input: n is the node to evaluate
Data: M is the causal matrix obtained during the process mining procedure
Output: A boolean that indicates if the node verifies the constraints of IMS LD for plays.

```
1  g ← the groups defined in n
2  for i = 0 to length(g) − 1 do
3      g_i ← g[i]
4      for j = 0 to length(g_i) do
5          a_j ← g_i[j]
6          for k = k + 1 to length(g) do
7              g_k ← g[k]
8              for l = 0 to length(g_k) do
9                  a_l ← g_k[l]
10                 if a_j and a_l are not in parallel in M then
11                     return false
```



(a) *Petri net*

Task	I(Task)	O(Task)
A	{}	{}
B	{}	{{C}}
C	{{B}}	{}

(b) *Causal matrix*

Figure 8: Petri net with three activities and its causal matrix

case, we just check *(i)* that a_j is not connected to a_l, *(ii)* that a_l is not connected to a_j, and *(iii)* that $a_j \neq a_l$.

Let suppose the Petri net depicted in Figure 8 and its corresponding causal matrix, and the previously described search space depicted in Figure 7. A correct ordering of the activities of the causal matrix would return $[A\ B\ C]$. Taking this into account, Algorithm 2 would process the nodes of the first level and determine that nodes 01 and 11 are not in conformity with the causal matrix. Specifically, in line 10 it would verify for both cases that task B is not in parallel with C.

7.3. Consistency of acts

Algorithm 3 checks the structural consistency at the level of acts. Specifically, this function is used to verify that the groups defined at this level are in conformity *(i)* with the IMS LD specification and *(ii)* with the causal matrix obtained from the process mining. In this case, IMS LD requires that the acts of a play must be in sequence. Therefore, the algorithm has an external loop that iterates the plays identified as consistent, and for each one of these plays it checks *(i)* that the activities of an act are isolated from other acts (lines 6-8), *(ii)* that last activities of an act and the next ones are not in conflict because of a choice node (lines 12-13), and *(iii)* that the acts are connected (lines 14-15), i.e., that the last activities of an act are connected to the first activities of the next act.

Continuing with the example depicted in Figure 8, let suppose that we want to evaluate the node 01 which father play node is also 01. In the two iterations of the first loop, $g_0 \leftarrow [A]$ and $g_1 \leftarrow [B\ C]$ will both have $S_o \leftarrow \{\}$ and

Algorithm 3: acts

Input: n is the node to evaluate
Data: M is the causal matrix obtained during the process mining procedure
Output: A boolean that indicates if the node verifies the constraints of IMS LD for acts.

```
1  foreach group k of the parent node of n do
2      g ← the groups defined in n that are contained in the father group k
3      for i = 0 to length(g) do
4          g_i ← g[i]
5          S_o ← set of final activities in g_i
6          foreach activity a in g_i \ S_o do
7              if a is connected to an activity not in g_i then
8                  return false
9          if i + 1 < length(g) then
10             g_j ← g[i + 1]
11             S_i ← set of initial activities in g_j
12             if there is a shared choice between the activities if S_o and S_i then
13                 return false
14             foreach activity a_o in S_o do
15                 foreach activity a_i in S_i do
16                     if a_i and a_o are not connected then
17                         return false
```

$S_i \leftarrow \{\}$ and thus verify the conditions of the acts' level. However, let consider that the node to evaluate is 01 and its parent play node is 00. For the case $g_0 \leftarrow [A\ B]$ the algorithm will fail in line 7 since B is connected to an activity that is not included in g_0.

7.4. Consistency of role-parts

In order to check that role-parts are correctly defined we must verify that:

- All the activities included in the role-part have the same role.
- The groups identified in the role part are in parallel.

Since the conditions are similar to those imposed to plays, we will not detail the algorithm used to verify the consistency of role-parts. For instance, the node 10 pointed by the parent node 00 is a valid configuration since the activity A is in parallel with the activities B and C. As aforementioned, all configurations that are valid at the play level are also valid at this level, but, in addition, nodes must also be compatible with its parent node. Therefore, if the parent node is 11, the node 10 would no longer be a valid configuration.

7.5. Consistency of activities

In IMS LD, activities can be simple or structured as sequences or selection of activities. Taking this into account, Algorithm 4 checks that all the groups verify one of these categories. Specifically, the function *activity* only checks

Algorithm 4: activities

Input: n is the node to evaluate
Data: M is the causal matrix obtained during the process mining procedure
Output: A boolean that indicates if the node verifies the UoL constraints of IMS LD.

```
1 foreach group k of the parent node of n do
2     g ← the groups defined in n that are contained in the father group k
3     foreach group g_i in g do
4         if not activity(g_i) or not sequence(g_i) or not selection(g_i) then
5             return false
```

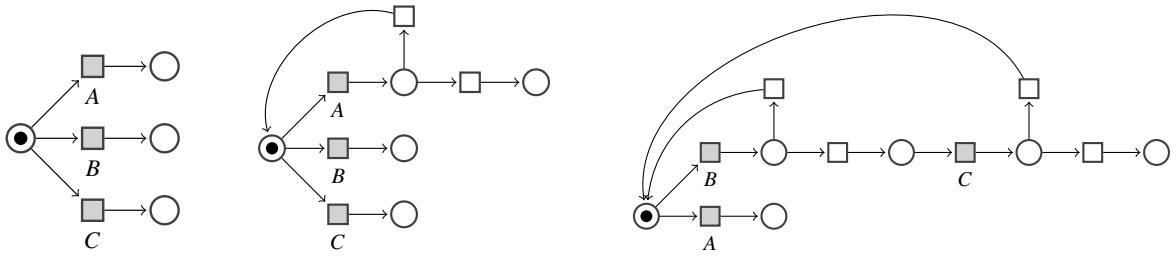
Algorithm 5: selection

Input: g is the group of activities to evaluate
Data: M is the causal matrix obtained during the process mining procedure
Output: A boolean that indicates if the group verifies the selection structure of IMS LD.

```
1 if length(g) < 2 then
2     return false
3 foreach activity a_i in g do
4     if a_i is not in a choice with another activity in g then
5         return false
6 if father node is a selection then
7     return false
```

that the group has q unique element. The function *sequence* verifies that the elements of the group are in sequence, in the same way as Algorithm 3 checks that all the acts of a play are in sequence. Finally, the function *selection* checks the last activity structure of IMS LD. Notice that selections are a complex structure since they combine two patterns, such as a choice between several activities (simple or complex), and a loop, so students may select the same activity more than once. The main issue here is that process mining algorithms are not able to detect this pattern as it is modelled in IMS LD [38]. Instead they usually detect partial choices and combine them with multiple loops. In order to identify this pattern from the logs we just need to detect if the activities are in a choice with any other activity in the selection since the loops spread the choice dependencies, lines 3-5 of Algorithm 5. Finally, we do not allow the concatenation of selections (lines 6-7) since they do not change the behaviour of the net and just complicate the design: a subselection can be moved to a father selection without changing the behaviour of the control construct.

For instance, Figure 9 represents different ways in which process mining may retrieve a selection of three activities. Figure 9a depicts the simplest selection pattern in which the three activities compete for the tokens of their input place and thus only one of them can be chosen. In Figure 9b, the first branch also includes a loop, and thus students can select more than one activity. Finally, Figure 9c combines a choice, a loop, and a sequence structure. Notice, that our algorithm detects both the three cases since activities A, B, and C share the same input.



(a) *Choice of activities*

(b) *Selection where the activities are in choice and in a loop*

(c) *Selection where one of the branches is a sequence of two activities with two loops*

Figure 9: Three different ways of representing a selection. Transitions colored in gray represent the activities of the UoL.

8. Results

The validation of the presented approach has been done with a set of UoLs with different degrees of complexity. In a first batch, four UoLs of the Degree of Computer Science, at the University of Santiago de Compostela, were performed in the real environment of OPENET4LD [38] by students. To complete the dataset, we selected another five UoLs, collected by the Open University in the Netherlands from different European projects[5], and simulated their behaviour in OPENET4LD environment but with virtual students.

All UoLs were generated in IMS LD, half of the units were performed in a real environment by students while the other half was simulated, and logs were recorded by OPENET4LD[6] [38]. The objective of this experiment is to recompile the UoLs in IMS LD format and minimize information loss. Table 1 lists the nine UoLs tested, on the basis of their activities, structures of activity, acts, plays, and properties, ranging from UoLs with one only act to UoLs with several acts and different activity structures:

- *AutomatonClass* is an adaptative UoL about Automata Theory and Formal Languages.
- *TALF* is another UoL on the topic of Automata Theory and Formal Languages.
- *Boeing* is an UoL about the safety regulations when removing certain parts of a Boeing engine (simulated).
- *Cam* specifies the activities and exercises needed to complete a lecture (simulated).
- *Driving* specifies the process of a driving school (simulated).
- *Programming* is about learning imperative programming.
- *POO* is a UoL about learning object oriented programming.

[5] `http://dspace.ou.nl/handle/1820/16/`

[6] Logs and images of the extracted Petri nets are available at `http://tec.citius.usc.es/SoftLearn/Compiling.html`

UoL	#LA	#ST	#AC	#PL	#RU	#PR	Activity structures			
							Sequence	Choice	Parallelism	Loop
PeerReview	14	7	2	1	10	15	✓	✓	✓	
Cam	10	2	3	1	7	13	✓	✓	✓	
POO	9	3	6	1	6	12	✓	✓	✓	✓
Driving	7	3	5	1	3	5	✓	✓		
Boeing	10	5	4	1	9	12	✓	✓	✓	
AddWork	7	2	3	1	3	8	✓	✓		
Automaton	9	2	5	1	7	9	✓	✓	✓	
TALF	11	5	7	1	8	9	✓	✓	✓	
Programming	4	1	2	1	2	4	✓	✓	✓	

#LA, #ST, #AC, #PL, #RU, and #PR stands for the number of activities, structures of activity, acts, plays, rules and properties, respectively.

Table 1: Structural features of the UoLs that have been used in the experiment.

	PeerReview	**Cam**	**POO**	**Driving**	**Boeing**	**AddWork**	**Automaton**	**TALF**	**Programming**
# instances	186	186	190	97	134	120	120	144	42

Table 2: Number of instances of each UoL.

- *PeerReview* explains the interaction in a peer review process (simulated).
- *AddWork* represents a quiz with different outcomes depending on the results (simulated).

Summarizing, four of the UoLs belong to courses of the Computer Science degree, while the remaining UoLs were taken from tutorials, seminars, and workshops previously recorded in OPENET4LD.

8.1. Process mining results

In order to check the efficiency of ProDiGen for learning flows discovery, we have conducted an experiment with the nine UoLs shown in Tab. 1 that have been undertaken in the OPENET4LD environment. OPENET4LD collects all the events generated by the learners when they perform the learning activities of an UoL, and then, with the OPENET4LD Adapter, we transform this behavior into a XES log. Although some of these UoLs are designed to be undertaken by several roles collaborating among them, ProDiGen discovers the learning flow related to each role. Table 2 shows the number of instances —traces— of each example. As we can see, only *Drive* and *Programming* UoL have less than 100 instances, i.e., students that have participated in the courses. The remaining units are in between 120 and 190 instances, which is enough to perform with a high degree of confidence the mining process —for both the models and the rules (Section 8.2).

	UoLs								
	PeerReview	**Cam**	**POO**	**Driving**	**Boeing**	**AddWork**	**Automaton**	**TALF**	**Programming**
C	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P	0.85	0.85	0.90	1.00	0.95	0.82	0.90	1.00	0.86
$(\#A, \#P, \#T)$	(15,7,6)	(38,13,13)	(36,12,18)	(26,10,11)	(24,10,10)	(36,15,16)	(34,12,17)	(34,12,17)	(37,13,16)

Table 3: Performance of the genetic algorithm for learning flow discovery.

The performance of ProDiGen over the UoLs has been measured taking into account completeness —fitness replay—, precision and simplicity with two metrics of the state of the art. For completeness, we use the proper completion measure [34], which is the fraction of properly completed process instances. Proper completion (C) takes a value of 1 if the mined model can process all the traces without having missing tokens or tokens left behind. On the other hand, to measure the simplicity, we used the alignment precision (P) defined in [2]. For the simplicity, we show the number of arcs (#A), places (#P) and transitions (#T) of the retrieved Petri net.

Table 3 shows the results of ProDiGen over the 8 UoLs. As can be seen, ProDiGen is able to retrieve, for all the cases, a model that perfectly fits the behavior of the recorded event logs, i.e., the fitness replay is equal to one in all the UoLs. Moreover the precision of the mined models is high, which indicates that the models do not underfit the log, i.e. they do not show much more behavior than the actually observed in the logs. Related to these precision levels, in none of the cases the retrieved model is the overfitted *trace model*, a solution that creates a path for each trace of the log, which can be seen with the simplicity measure —the number of actual transitions and arcs.

8.2. Rules mining results

A ten fold cross-validation has been performed for each one of the rules of each UoL detailed in Table 1. Table 4 details the results of the adaptive rules mining. We can see that, in 54.54% of the cases, the rule did classify correctly the 100% of the instances. In fact, the results are quite good considering that in 77.27% of the cases at least 90% of instances were correctly categorized, percentage that even grows to 90.91% if we consider the cut level of 80%. Only four cases obtain less positive results. Specifically, three of them are in the *Boeing* UoL and one in the *PeerReview* UoL. For instance, one of the rules of the *Boeing* UoL in this situation is defined as follows:

```
visible_Test_components = false: false
visible_Test_components = true
|   visible_Test_hazard = false: true
|   visible_Test_hazard = true
|   |   Lessons_components_counter <= 1
|   |   |   visible_Extra_Lessons_hazards = false
|   |   |   visible_Extra_Lessons_hazards = true
|   |   Lessons_components_counter > 1: false
```

Rule id	Correctly
PeerReview	
1	100%
2	86.02%
3	100%
4	98.39%
5	88.07%
6	95.70%
7	100%
8	53.22%
9	100%
10	95.70%

Rule id	Correctly
Cam	
1	100%
2	100%
3	98.82%
4	97.93%
POO	
1	100%
2	99.47%
3	100%
4	100%
5	100%

Rule id	Correctly
Driving	
1	97.93%
2	90.72%
3	86.60%
Boeing	
1	89.55%
2	74.62%
3	95.52%
4	79.10%
5	72.39%
6	88.06%

Rule id	Correctly
AddWork	
1	100%
2	100%
Automaton	
1	99.17%
2	100%
3	100%
4	100%
5	100%
6	100%
7	100%

Rule id	Correctly
TALF	
1	100%
2	100%
3	100%
4	100%
5	100%
6	100%
7	86.81%
Programming	
1	100%
2	100%

Table 4: Performance of the adaptive rules mining, where the percentage shows the instances that have been correctly classified by the rule learned by the decision tree.

As we can see, this rule uses four different variables to correctly classify 72.39% of the instances. In this and the other cases in which the learning process obtained weaker results, the number of examples were clearly insufficient to get the convergence of the algorithm. However, it would be misleading to point out that the number of variables is the unique factor, since most of the other units have rules with a similar number of variables. In these cases, we concluded that the complexity of the rule has more influence and would require a greater number of examples to obtain better results.

Finally, it should be mentioned that learning these rules is usually a difficult task, since the criteria used for the adaptation are not always clear. For instance, in many cases the instructor does manually the adaptation and not always with a uniform criteria. However, in these cases rules mining may achieve a secondary objective since it will precisely clarify these criteria from the event logs.

8.3. Reenginering results

Table 5 shows the results of the reengineering part of our proposal. As we can see, the processes were recompiled successfully for all UoLs. It should be noted the huge number of solutions that can be derived from the causal matrix in some of the cases. For instance, there are 394 different possibilities to structure a valid UoL from *AddWork* logs. There is a simple explanation: IMS LD is mainly structured with parallels and sequences. Plays and role-parts are parallel structures, while acts and sequences of activities are an ordered list of elements. Therefore, the number of possible combinations grows as the number of parallels and sequences increases. Notice that there would be even more solutions if we did not limit the concatenation of selections between a father and a child node.

The time required to obtain all the results depends on two factors. On the one hand, the net complexity, i.e., the greater is the number of structures, the more combinations. On the other hand, the structures of activities play an

	PeerReview	Cam	POO	Driving	Boeing	AddWork	Automaton	TALF	Programming
# solutions	2	12	9	90	2	394	44	55	2
Time (ms)	184531	10960	29476	603	1162	2273	7003	65335	144

Table 5: Results of the reengineering process

important role in computational time since they increase the depth of the search tree. Therefore, it is not surprising that *PeerReview* and *TALF* UoLs need more time since the have more structural elements and activities.

It should be noted that, from a practical viewpoint, it does not make sense to show all the reengineering results to an instructor. For instance, the selection of the most suitable structure from the 394 solutions returned for the AddWork UoL would be unmanageable for an instructor without a proper way to filter the results. For this reason, solutions are ordered according to criteria identified in [11]. In this study, participants were asked to transform a given textual design description into an IMS LD UoL. The analyses identified a number of conceptual structures which presented challenges to teachers' understanding, e.g., the management of role-parts. Specifically, in our proposal we use the following ordering criterion:

- *Simplicity*. UoLs with fewer structural elements take precedence.
- *Plays precedence*. Plays and role-parts are used to define parallel structures within a UoL, but plays have precedence over role-parts.
- *Acts precedence*. Acts and activity-structures are used to define sequences of activities within a UoL, but acts have precedence over activity-structures.

9. Conclusions and Future Work

In this paper we have proposed a global approach that facilitates the reuse of UoLs defined in legacy systems or VLEs. Our solution has been implemented to support the mining of event log files (i) to obtain the learning flow, formalized as a Petri net, and performed by students and instructors, and (ii) to identify the adaptation rules, represented as decision trees, used to tailor the learning flow to each student. An important feature of the described approach is its independence from any target EML, although in this paper the reengineering part of the framework is tied to IMS LD. However, it is sufficient to define the translation, from the Petri net models to the specific process representation, and from the decision tree to the rules grammar used by the EML, to have a compete specification of the UoL. In fact, as future work we plan to use this framework to recompile the same event logs to a different EML, such as ADL SCORM.

To validate this approach, we tested our framework with 9 UoLs with different degrees of complexity. Specifically, the three parts of the reengineering approach have been analyzed separately. Firstly, we showed that process mining

is a good solution to retrieve a process structure from a set of event log files. Specifically, ProDiGen had a precision greater than 90% in most of the cases, and where the worst precision was of 82%. These high precision values minimize the number of UoLs obtained by the reengineering process. Secondly, the identification of the adaptive rules also obtained very good results. In fact, the exact rule was extracted in most of the analyzed cases and at least 80% of the instances correctly classified in nearly all the remaining cases. Finally, we must mention that the correct IMS LD structure was retrieved for all the analyzed UoLs.

It should be remarked that a secondary objective of this paper is to facilitate the reuse of UoLs but from the perspective of instructors. In this sense, our system only requires the participation of instructors to select the most suitable process structure from the set of solutions recompiled by our framework. Notice that this is still a difficult task, since instructors do not always have a deep knowledge of how IMS LD structures a UoL. This decision is more complex as the number of recompiled structures, that match the logs, grows. Therefore, as future work we plan to define some criteria, in addition to those identified in [11], to order the recompiled UoLs, making this selection easier. So far our experience makes us think that the complexity is in the activities part of the UoL, since plays, acts, and role-parts are usually not considered when designing a UoL in non-IMS LD environments. However, this point must still be verified.

Acknowledgments

This work was supported by the Spanish Ministry of Economy and Competitiveness under the project TIN2014-56633-C3-1-R and by the European Regional Development Fund (ERDF/FEDER) under the project CN2012/151 of the Galician Ministry of Education.

[1] van der Aalst, W.M.P., 2011. Process Mining: Discovery, Conformance and Enhancement of Business Processes. 1st ed., Springer Publishing Company, Incorporated.

[2] van der Aalst, W.M.P., Adriansyah, A., van Dongen, B., 2012. Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2, 182–192.

[3] van der Aalst, W.M.P., Günther, C.W., 2007. Finding structure in unstructured processes: The case for process mining, in: Basten, T., Juhás, G., Shukla, S.K. (Eds.), Proceedings of the 7th International Conference on Application of Concurrency to System Design, (ACSD), Bratislava, Slovak Republic. pp. 3–12.

[4] van der Aalst, W.M.P., Weijters, T., Maruster, L., 2004. Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering. 16, 1128–1142.

[5] Abdous, M., 2011. Towards a framework for business process reengineering in higher education. Journal of Higher Education Policy and Management 33, 427–433.

[6] Abdous, M., He, W., 2008. A framework for process reengineering in higher education: A case study of distance learning exam scheduling and distribution. The International Review of Research in Open and Distributed Learning 9.

[7] Bergenthum, R., Desel, J., Harrer, A., Mauser, S., 2012. Transactions on Petri Nets and Other Models of Concurrency V. Springer-Verlag, Berlin, Heidelberg. chapter Modeling and Mining of Learnflows. pp. 22–50. URL: `http://dl.acm.org/citation.cfm?id=2231056.2231058`.

[8] Global Learning Consortium, I., 2003. IMS Learning Design Information Model. URL: `http://www.imsglobal.org/learningdesign/index.html`. Version 1.0 Final Specification.

[9] Dalziel, J., 2003. Implementing learning design. the learning activity management system (lams), in: Proceedings of the 20th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education (ASCILITE), IADIS Press, Adelaide, Australia. pp. 51–58.

[10] Delias, P., Doumpos, M., Grigoroudis, E., Manolitzas, P., Matsatsinis, N.F., 2015. Supporting healthcare management decisions via robust clustering of event logs. Knowledge-Based Systems 84, 203–213.

[11] Derntl, M., Neumann, S., Griffiths, D., Oberhuemer, P., 2012. The conceptual structure of ims learning design does not impede its use for authoring. IEEE Transactions on Learning Technologies 5, 74–86.

[12] Dodero, J.M., del Val, Á.M., Torres, J., 2010. An extensible approach to visually editing adaptive learning activities and designs based on services. Journal of Visual Languages & Computing 21, 332–346.

[13] van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P., 2005. The prom framework: A new era in process mining tool support, in: Ciardo, G., Darondeau, P. (Eds.), Proceedings of the 26th International Conference Applications and Theory of Petri Nets, (ICATPN), Miami, USA. pp. 444–454.

[14] Griffiths, D., Beauvoir, P., Sharples, P., 2008. Advances in editors for IMS LD in the tencompetence project, in: Proceedings of the 8th IEEE International Conference on Advanced Learning Technologies, (ICALT), Santander, Cantabria, Spain. pp. 1045–1047.

[15] Hernández-leo, D., Villasclaras-Fernández, E.D., Asensio-Pérez, J.I., Dimitriadis, Y., Jorrín-Abellán, I.M., Ruiz-Requies, I., Rubia-Avi, B., 2006. Collage: A collaborative learning design editor based on patterns. Educational Technology & Society 1, 58–71.

[16] Heyer, S., Oberhuemer, P., Zander, S., Prenner, P., 2007. Making sense of IMS learning design level B: from specification to intuitive modeling software, in: Duval, E., Klamma, R., Wolpers, M. (Eds.), Proceedings of the 2nd European Conference on Technology Enhanced Learning, (EC-TEL), Crete, Greece. pp. 86–100.

[17] Howard, L., Johnson, J., Neitzel, C., 2010. Examining learner control in a structured inquiry cycle using process mining, in: de Baker, R.S.J., Merceron, A., Jr., P.I.P. (Eds.), Proceedings of the 3rd International Conference on Educational Data Mining, (EDM), Pittsburgh, PA, USA. pp. 71–80.

[18] Karagiorgi, Y., Symeou, L., 2005. Translating constructivism into instructional design: Potential and limitations. Educational Technology & Society 8, 17–27.

[19] Karampiperis, P., Sampson, D., 2004. A flexible authoring tool supporting adaptive learning activities, in: Proceedings of the IADIS International Conference on Cognition and Exploratory Learning in Digital Age (CELDA), IADIS Press, Lisbon, Portugal. pp. 51–58.

[20] Karray, M., Chebel-Morello, B., Zerhouni, N., 2014. PETRA: process evolution using a trace-based system on a maintenance platform. Knowledge-Based Systems 68, 21–39.

[21] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P., 2013. Discovering block-structured process models from event logs containing infrequent behaviour, in: Lohmann, N., Song, M., Wohed, P. (Eds.), Business Process Management Workshops - BPM, pp. 66–78.

[22] de Leoni, M., van der Aalst, W.M.P., Dees, M., 2014. A general framework for correlating business process characteristics, in: Sadiq, S.W., Soffer, P., Völzer, H. (Eds.), Proceedings of the 12th International Conference on Business Process Management, (BPM), Haifa, Israel. pp. 250–266.

[23] Liu, T., Cheng, Y., Ni, Z., 2012. Mining event logs to support workflow resource allocation. Knowledge-Based Systems 35, 320–331.

[24] Martens, H., Vogten, H., 2009. Coppercore 3.3.

[25] Martinez-Ortiz, I., Sierra, J.L., Fernandez-Manjon, B., 2009. Authoring and reengineering of ims learning design units of learning. IEEE Transactions on Learning Technologies 2, 189–202.

[26] de Medeiros, A., 2006. Genetic Process Mining. Ph.D. thesis. Technische Universiteit Eindhoven.

[27] Paquette, G., Léonard, M., 2006. The educational modeling of a collaborative game using MOT+LD, in: Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies - ICALT, Kerkrade, The Netherlands. pp. 1156–1157.

[28] Pechenizkiy, M., Trcka, N., Vasilyeva, E., van der Aalst, W.M.P., Bra, P.D., 2009. Process mining online assessment data, in: Barnes, T., Desmarais, M.C., Romero, C., Ventura, S. (Eds.), Proceedings of the 2nd International Conference on Educational Data Mining, (EDM), Cordoba, Spain. pp. 279–288.

[29] Poncin, W., Serebrenik, A., van den Brand, M., 2011a. Mining student capstone projects with FRASR and prom, in: Lopes, C.V., Fisher, K. (Eds.), Proceedings of the 26th Annual international conference companion on Object-Oriented Programming, Systems, Languages, and Applications, (OOPSLA), Portland, OR, USA. pp. 87–96.

[30] Poncin, W., Serebrenik, A., van den Brand, M., 2011b. Process mining software repositories, in: Mens, T., Kanellopoulos, Y., Winter, A. (Eds.), Proceedings of the 15th European Conference on Software Maintenance and Reengineering, (CSMR), Oldenburg, Germany. pp. 5–14.

[31] Quinlan, J.R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[32] Rodríguez, M.C., Derntl, M., Botturi, L., 2010. Visual instructional design languages. Journal of Visual Languages and Computing 21, 311–312.

[33] Romero, C., Ventura, S., García, E., 2008. Data mining in course management systems: Moodle case study and tutorial. Computers & Education 51, 368–384.

[34] Rozinat, A., van der Aalst, W.M.P., 2008. Conformance checking of processes based on monitoring real behavior. Information Systems 33, 64–95.

[35] Rozinat, A., de Medeiros, A.K.A., Günther, C.W., Weijters, A.J.M.M., van der Aalst, W.M.P., 2007. The need for a process mining evaluation framework in research and practice, in: ter Hofstede, A.H.M., Benatallah, B., Paik, H. (Eds.), Proceedings of the 5th International Conference on Business Process Management, (BPM), Brisbane, Australia. pp. 84–89.

[36] Vázquez-Barreiros, B., Mucientes, M., Lama, M., 2015. ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm. Information Sciences 294, 315–333.

[37] Verbeek, H.M.W., Buijs, J., Van Dongen, B.F., Van Der Aalst, W.M.P., 2011. Xes, xesame, and prom 6, in: Soffer, P., Proper, E. (Eds.), Information Systems Evolution. Springer Berlin Heidelberg. volume 72 of *Lecture Notes in Business Information Processing*, pp. 60–75.

[38] Vidal, J.C., Lama, M., Bugarín, A., 2012. Petri net-based engine for adaptive learning. Expert Systems With Applications 39, 12799–12813.

[39] Weijters, A., Ribeiro, J., 2011. Flexible heuristics miner (fhm), in: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011), IEEE. pp. 310–317.

[40] van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A., 2009. Process discovery using integer linear programming. Fundam. Inform. 94, 387–412.