Reconstructing IMS LD Units of Learning from Event Logs

Juan C. Vidal, Manuel Lama, Borja Vázquez, and Manuel Mucientes

Research Center on Information Technology (CiTIUS),
University of Santiago de Compostela
{juan.vidal,manuel.lama,borja.vazquez,manuel.mucientes}@usc.es

Abstract. In this paper a novel approach to facilitate the reuse of units of learning (UoLs) is presented. Typically, e-learning platforms do not provide the means to retrieve designed UoLs in a standardized format to be reused in a different platform, but they have in common that the students and teachers interaction with the system is logged to files. Taking this into account, we propose to use these logs and apply a three steps re-engineering approach to translate these UoLs into an accepted educational modelling language, specifically IMS LD. In the first step, the sequence of activities and their functional dependencies are learned by a process mining algorithm. In the second step, another algorithm analyses the variables and their value change in order to learn the adaptation rules that may have been defined in the UoL. And finally, in the last step the inferred process structure and rules are matched with the typical structure of activities, acts, and plays defined by IMS LD.

Keywords: IMS LD, process mining, adaptive rules mining, learning analytics, learning flows reengineering.

1 Introduction

In the last decade, an important effort to develop Educational Modelling Languages (EMLs) has been made. The aim of these languages is to describe from a pedagogic point of view the learning design of a course: that is, the flow of learning activities undertaken by the learners to achieve the objectives of a course using educational content and services. From these EMLs, the IMS Learning Design specification [3] has emerged as the *de facto* standard for representing learning designs that can be based on a wide range of pedagogical techniques. Although there is some controversy about whether IMS LD is too complex to be understood by teachers from a practical point of view [5], most of authors highlights this complexity as a barrier for the adoption of IMS LD [8].

To deal with this issue a number of user-friendly authoring tools have appeared. These tools provide graphical interfaces that allow to hide the complexity of the IMS LD control structure and adaptive components [7]; offer visual templates or patterns for creating learning designs based on pedagogical strategies such as collaborative learning or project based learning [6]; or translate the

C. Rensing et al. (Eds.): EC-TEL 2014, LNCS 8719, pp. 345–358, 2014.

learning design to the IMS LD specification from authoring tools that are not based on IMS LD [4]. However, even with these tools the authoring process of IMS LD units of learning (UoLs) is not easy for teachers when the UoLs are complex or require to use advanced features of IMS LD.

The automatic reconstruction of UoLs could relieve this issue. In [8] authors present an approach that focus on the reegineering 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. The main drawback of this approach is that the UoL is not automatically reconstructed from the scratch and needs the supervision of teachers and even developers. Other similar approaches have been proposed in the literature [2,11], although not for IMS LD.

In this paper, we present an approach to *automatically* reconstruct the IMS LD representation of a UoL from the events generated by the learners in the virtual learning environment. This objective is achieved in three different steps. Firstly, the learning flow of the UoL is automatically extracted through a *process discovery algorithm* [1] which guarantees the completeness of the discovered learning flow. 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 from the event logs by a *decision tree* learning algorithm and integrated in the IMS LD structure.

The paper is structured as follows: in Section 2 we present the framework that supports the mining of log files and the reconstruction of IMS LD. In sections 3 and 4 we precisely detail the how the learning flow and adaptive rules are mined from the logs, respectively, while in Section 5 we close the circle and describe how we transform these two models to IMS LD. Finally, in Section 6 we present the conclusions of the work.

2 Framework

Fig. 1 depicts the conceptual framework for reconstructing IMS LD UoL from the events generated by learners. The framework is composed by the following components:

- *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.
- *Virtual Learning Environment*. From an educational point of view, virtual learning environments (VLEs) provide the means to carry out the learning activities planned for a UoL, allowing learners to access to the learning contents and executing the services required to facilitate those activities such

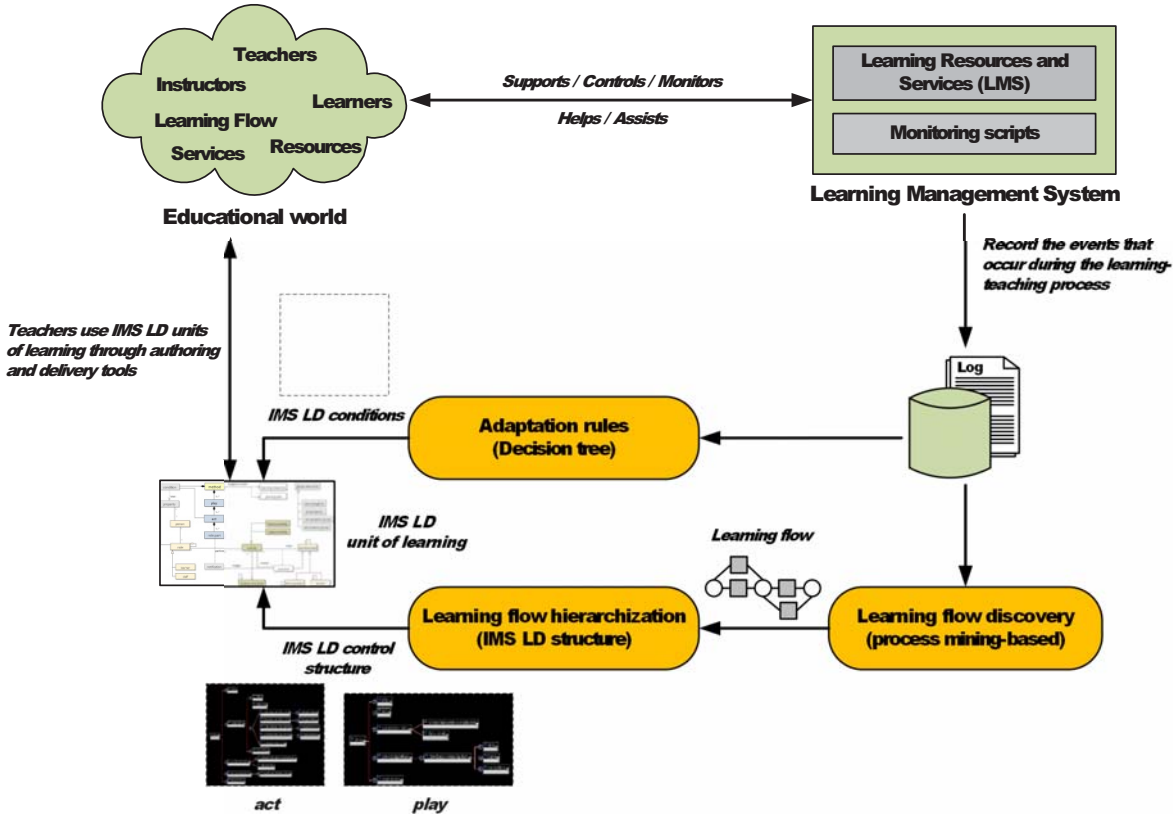

Fig. 1. Framework for IMS LD UoLs reconstruction

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.

- *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.
- *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 do apply the knowledge about the IMS LD control structure to create activity structures, acts or plays.
- *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.

In the following sections we will detail the last three components that constitute the core of this framework.

3 Mining the Learning Flow from Event Logs

The aim of the process discovery algorithm (PDA) [1] is to identify the workflow that represents the learning flow followed by the learners during a UoL. To achieve this objective, the PDA will only needs to process the metadata information provided by the event logs. From the perspective of process mining, the quality of a process discovery algorithm is measured taking into account the following metrics:

- *Completeness*, which indicates how much of the behaviour observed in the event log can be reproduced by the discovered learning flow. Thus, a discovered learning flow is complete when it can reproduce all the events contained in the log database. In order to guarantee correct reconstructions of IMS LD UoLs, all the activities undertaken by learners have to be included in the mined learning flow. Therefore, the completeness of discovered learning flows is a *hard requirement* for the PDA.
- *Precision*, which measures if the discovered learning flow is overly general, allowing an additional behaviour that is not represented in the log. Thus a discovered learning flow is precise when it cannot reproduce event traces that are not available in the log database. From the point of view of the IMS LD reconstruction to discover precise learning flows is not a requirement as hard as completeness, since the extra behaviour has not been undertaken by any learner.
- *Minimality*, which refers to discovered learning flows with the minimal structure that reflects the behaviour contained in the log database. A desirable requirement for the PDA is to obtain simple learning flows, since the simpler is the discovered learning flow, the easier is to reconstruct the IMS LD control structure.

Taking these measures into account, we have developed a genetic algorithm that discovers complete learning paths with very high values for precision and simplicity. Algorithm 1 describes this algorithm, where the first three steps correspond to its initialization with t representing the number of iterations of the algorithm, *timesRun* is used to detect situations in which the search gets stuck, and restart counts the number of times the algorithm is reinitialized. The execution cycle of the genetic algorithm is as follows:

- A population is created with a group of individuals where each individual is a potential solution, i.e., a learning flow. In this algorithm, learning flows are

ALGORITHM 1. Genetic algorithm for discovery of learning flows

```
Initialize population
Evaluate population
t = 1, timesRun = initialTimesRun, restarts = 0
while t ≤ maxGenerations && restarts < maxRestarts do
    Selection
    Crossover
    Mutation
    Evaluate new individuals
    Replace population
    t = t + 1
    if bestInd(t) == bestInd(t − 1) then
        timesRun = timesRun − 1
    end
    if none of the individuals of the population have been replaced then
        timesRun = timesRun − 1
    end
    if timesRun < 0 then
        Reinitialize population
        Evaluate population
        timesRun = initialTimesRun, restarts = restarts + 1
    end
end
```

represented through causal matrices which can be easily translated into Petri nets [9]. Fig. 2 depicts the Petri that models a UoL where the pedagogical objective is to learn about polymorphism in object-oriented programming.

- These individuals are then evaluated with a fitness measure that indicates how well each individual is able to reproduce the behaviour shown in the log database.
- After this evaluation process, the population evolves by selecting those individuals with a higher fitness. Then, it generates new individuals through genetic operators like crossover, which combines two individuals, and mutation, which modifies an individual.
- These steps will be repeated in a cycle until any of the termination conditions, *maxGenerations* and *maxRestarts*, are fullfilled.

In this genetic algorithm, the fitness measure (F) for an individual, i.e., for a learning flow, is based on its completeness (C_f), precision (P_f), and simplicity (S_f). When two individuals are compared to decide which of them will be selected, these three measures are considered separately:

$$\begin{aligned} F(a) > F(b) \iff & \{C_f(a) > C_f(b)\} \vee \{C_f(a) = C_f(b) \wedge P_f(a) > P_f(b)\} \\ & \vee \{C_f(a) = C_f(b) \wedge P_f(a) = P_f(b) \wedge S_f(a) > S_f(b)\} \end{aligned} \quad (1)$$

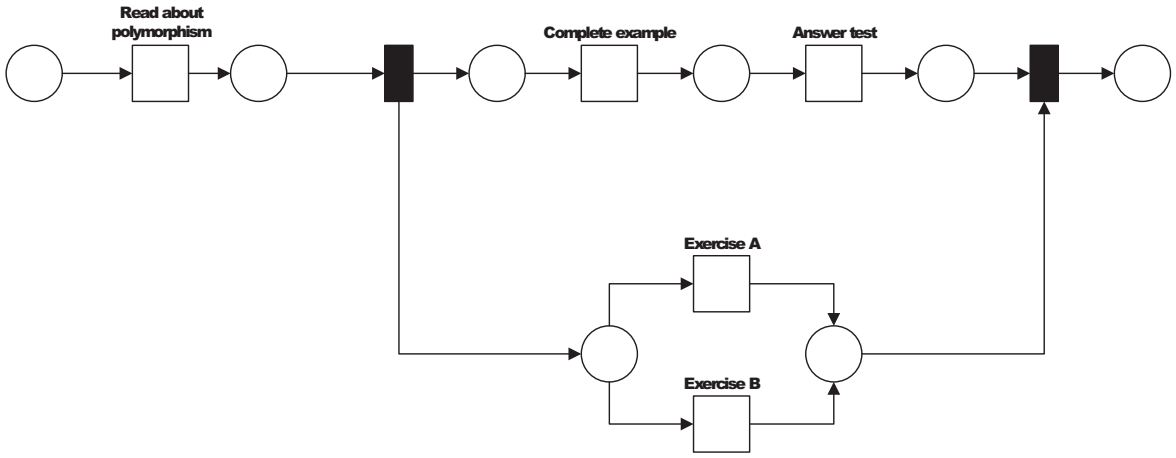

Fig. 2. Petri net that represents the learning flow of a UoL about polimorphism

Thus the individual with highest completeness will be selected. If the values for this measure are equal for both solutions, then the second measure to be compared is the precision. If both individuals have the same precision, the simplicity will be checked. Therefore, we are looking for the simplest learning flow from precise learning flows that are complete.

In order to check the efficiency of this algorithm for discovery learning flows, we have conducted an experiment with 14 UoLs[1] that have been undertaken in the OPENET4LD environment [12]. OPENET4LD collects all the events generated by the learners when they perform the learning activities of a UoL. Table 1 presents the results of the comparison between our genetic algorithm (GA) and two of the most used process discovery algorithms, alpha++ [14] and heuristic miner [13]. The results shows that only GA is able to discover learning paths that are complete and with high precision.

4 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 material 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 Fig. 3. This 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 ran the activity, the level of IMS LD of this activity (play, act, role-part), the execution time,

[1] Available at `http://dspace.ou.nl`

Table 1. Performance of the genetic algorithm for learning flow discovery (GA)

		Candidas	Introduction	IMS Learning	CLFriday	Tai Chi	endolab	Learning LMS	A2	Your Opinion	Caminatas	Trip around Spain	Debate	D2	A8
GA	P_f	1.0	1.0	0.95	1.0	1.0	1.0	1.0	1.0	1.0	0.77	1.0	1.0	1.0	1.0
	C_f	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	S_f	0.34	0.35	0.29	0.34	0.34	0.36	0.36	0.35	0.37	0.3	0.35	0.32	0.31	0.29
HM	P_f	1.0	1.0	0.96	1.0	1.0	1.0	1.0	1.0	1.0	0.73	1.0	1.0	1.0	1.0
	C_f	1.0	1.0	0.75	1.0	1.0	1.0	1.0	1.0	1.0	0.2	1.0	1.0	1.0	1.0
	S_f	0.34	0.35	0.29	0.34	0.34	0.36	0.36	0.35	0.37	0.28	0.35	0.32	0.31	0.29
α^{++}	P_f	1.0	1.0	0.94	1.0	1.0	1.0	1.0	1.0	1.0	0.73	1.0	1.0	1.0	1.0
	C_f	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	1.0	1.0	1.0	1.0
	S_f	0.34	0.35	0.31	0.34	0.34	0.36	0.36	0.35	0.37	0.29	0.35	0.32	0.31	0.29

(3, 'UOLID_201', 'SOID_0', 'GeoQuiz3', 'root', '2013-10-14 01:44:51', 'play', 'play-13193839-3ddf-41e2-a4b8-7af8dc13d059', 'play__0', 'pc_split'),
(4, 'UOLID_201', 'SOID_0', 'GeoQuiz3', 'root', '2013-10-14 01:44:52', 'act', 'act-e4901d0f-8232-4cce-a166-20e29133d279', 'act__0', 'ac_split'),
(5, 'UOLID_201', 'SOID_0', 'GeoQuiz3', 'root', '2013-10-14 01:44:52', 'act', 'act-e4901d0f-8232-4cce-a166-20e29133d279', 'act__0', 'as_split'),
(6, 'UOLID_201', 'SOID_0', 'GeoQuiz3', 'd9', '2013-10-14 01:44:52', 'role_part_role_part', 'act-e4901d0f-8232-4cce-a166-20e29133d279', 'role_part_role_part__0', 'as_start'),
...
(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'), ...

Fig. 3. Example of a text-based log file

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.

4.1 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 Fig. 3. 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.

4.2 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 Fig. 3 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.

4.3 Learning Rules with a Decision Tree

The identification of the adaptive rules is performed by means of J48 decision tree algorithm [10]. We selected this type of algorithms because of its simplicity, performance, and because an adaptive rule is very similar to a decision tree. As it is depicted in the left part of Fig. 4, 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 root to 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 Fig. 4 show 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.

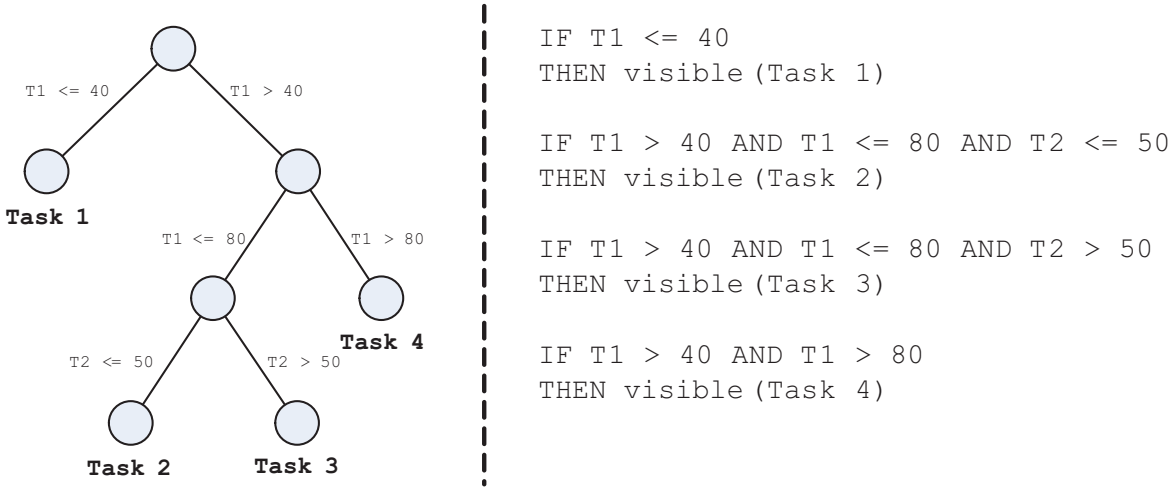

Fig. 4. Transformation of a decision tree to a DNF rule base

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 may reduce the accuracy of the tree. Moreover, the number of data required to learn increases with the number of variables. However, adaptive rule 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 which value has changed in the execution of the activity are included.

5 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 paper, two learning algorithms, presented in the former sections, address the first two levels of IMS LD. Specifically, the process structure of the learning is determined by a process mining approach and the adaptation mechanisms by a decision tree. However, the results obtained by these algorithms must be transformed to adapt to IMS LD characteristics.

5.1 A learning Flow as a Theatre Metaphor

Particularly, the IMS LD learning flow description of IMS LD is described as a theatre metaphor where there is a number of plays, that can be interpreted as runscripts that are *concurrently* executed, being independent of each other. Each of these plays consist of a set of acts, which can be understood as a module or chapter in a course. These acts are performed in *sequence*, and in each of them the participants in the UoL carry out *in parallel* an activity or a structure of

ALGORITHM 2. Identification of IMS LD activity structures of type *sequence*

Input: $G = (T, P, F)$ is the Petri net structure, $T = A \cup C$ is the set of transitions of the net where A is set of activities and C the set of control transitions, P is the set of places, $F \subseteq A \times P \cup P \times A$ is the set of arcs between transitions and places, R is the set of roles, U is the set of participants, $U_R \subseteq P \times R$ is set of user roles, $U_A \subseteq P \times A$ is the set of activities done by a user.

Output: SEQ is set of activity structures of type *sequence*.

```
1  A_R ← {(a, r) | a ∈ A ∧ r ∈ R ∧ (∀p | u ∈ U ∧ (u, r) ∈ U_R ∧ (u, a) ∈ U_A)}
2  SEQ ← ∅
3  for each a_i ∈ A do
4  |  S_i ← ∅
5  |  for each a_j ∈ A do
6  |  |  if (∃a_i, a_j, a_k, p | (a_i, p) ∈ F ∧ (p, a_j) ∈ F ∧ (p, a_k) ∈ F → a_j = a_k) then
7  |  |  |  if ∀p_a, p_b | (p_a, a_i) ∈ P_A ∧ (p_b, a_j) ∈ P_A ∧ p_a ≠ p_b → (∃r | r ∈
   |  |  |  R ∧ (p_a, r) ∈ P_R ∧ (p_b, r) ∈ P_R) then
8  |  |  |  |  S_i ← ∅
9  |  |  |  |  if ∃j | S_j ∈ SEQ ∧ a_i ∈ S_j then
10 |  |  |  |  |  S_i ← S_j
11 |  |  |  |  else
12 |  |  |  |  |  SEQ ← SEQ ∪ S_i
13 |  |  |  |  end
14 |  |  |  |  S_i ← S_i ∪ {a_i} ∪ {a_j}
15 |  |  |  end
16 |  |  end
17 |  end
18 end
```

activities, the which are executed in *sequence* or by *selecting* a specific number of them activities.

The problem therefore consists in identifying the different IMS LD elements in the Petri net defined by the process mining algorithm. Algorithm 2 shows one of the procedures defined for such purpose. Specifically, this algorithm is designed to identify the activity structures of type sequence in the Petri net $G = (T, P, F)$ as described in Section 3. Notice that the set T is composed of two types of transitions, namely activities and control structures, where te activities represent the learning and support activities of the learning process. The identification of sequences of activities takes into account (line 6) *(i)* that there is a direct dependency, that is, an arc between an activity a_i to a place p and also from p to a second activity a_j and that here is no other dependency between a_i and another activity a_k. Moreover, the two activities must also verify that they share the same role (line 7), otherwise they could be in different role-part structures.

The remaining algorithms used to identify the plays, acts, role-parts, or selections of activities use a very similar approach that consists in identifying the inner structural characteristics of each component. Specifically:

- *Selection of activities*. Very similar to Algorithm 2 where all the activities of the structure must share a role but also requires the activities in parallel between an *OR-split* and an *OR-join*.
- *Role-part*. The most difficult structure to identify. A role-part can be simple (just one activity with a specific role) or complex if it is composed by structures of activities, where sequences and selections can also contain sub-activities and structures.
- *Act*. Role-parts must match an *AND-JOIN* workflow pattern, that is, role-parts are split in parallel by an *AND-split* control structure and their execution is synchronized by an *AND-join*.
- *Play*. Defined by a set of acts in sequence.
- *Method*. Defined by a set of plays in parallel in an *AND-JOIN* workflow pattern.

5.2 Knowledge-Base of Adaptive Rules

IMS LD adapts the learning flow of a student according to a set of predefined properties and rules. Properties can have different scopes, such as global or local, different targets, such as a person or people with a specific role, or even the combination of different scopes and targets, such as, e.g., global personal or local role properties. The state of a learning process is consequently defined by these variables, and therefore, depending on their values the learning flow may vary according to students' needs. Usually, the state is composed by user properties and the presentation to a student of educational material is guided by his scores, his responses to questions, or, e.g., the time needed to solve a problem. However, properties with a global scope or targeted to roles (not just to individuals) provide to IMS LD the means to adapt collaborative or group-focused activities.

Taking this into account, the variables extracted from the event logs must be classified according to IMS LD types. The procedure detailed in Algorithm 3 returns the IMS LD scope and target of a property p previously identified in the log files. Firstly, it defines the time intervals (t, t_{+1}) between two events that have changed the value of p. For each one of these intervals, the algorithm identifies the value that the property had in this instant in the log files. Specifically, we define this value for each UoL and participant (line 9). Notice that we do not have always an event from which we can obtain the value of a property in a time interval. Therefore, we store the historical values for each UoL and participant during the processing of the log files and return the last known value of the property when no event was produced in the given time interval. The property target is determined in the *if-then-else* from lines 13 to 18. In this structure we identify the cases in which the property takes the same values for each time interval. If all the participants with the same role verify this predicate then the property target is set to *role*. Notice that since this structure is in a loop all the time intervals must verify this condition to finally be a *role*-targeted property. The verification of the scope is in between lines 21 to 26. In this *if-then-else* structure we just check that the value of a property is shared between all the

ALGORITHM 3. Classification of a variable identified in the event logs

Input: p is the property we want to classify, L is set of logs, M is the set of participants, U is the set of UoLs.
Output: $s_p \in S$ and $t_p \in T$ are the scope and target of p, respectively.
Data: $S = \{local, global\}$ and $T = \{role, personal\}$ are the set of scopes and targets defined in IMS LD, respectively.

$t \leftarrow 0$
$V \leftarrow \emptyset$
repeat
  $t_{+1} \leftarrow$ next time p value changed in L starting from t
  $V \leftarrow \emptyset$
  for $u \in U$ **do**
    $V_u \leftarrow \emptyset$
    for $m \in M$ **do**
      $V_{u,m} \leftarrow \{p$ values in UoL u for the user m in the interval $(t, t_{+1})\}$
      $V_u \leftarrow V_u \cup \{V_{u,m}\}$
    end
    $V_u \leftarrow V_u \cup \{V_{u,m}\}$
    if $\forall m_1, m_2, r, x \mid m_1, m_2 \in M \wedge m_1 \neq m_2 \wedge V_{u,m_1}, V_{u,m_2} \in V_u \wedge r \in role(m_1) \wedge r \in role(m_2) \wedge x \in V_{u,m_1} \rightarrow x \in V_{u,m_2}$ **then**
      if $t_p \neq personal$ **then**
        $t_p \leftarrow role$
      end
    else
      $t_p \leftarrow personal$
    end
    $V \leftarrow V \cup V_u$
  end
  if $\forall u_1, u_2, m_1, m_2, x \mid u_1, u_2 \in U \wedge u_1 \neq u_2 \wedge m_1, m_2 \in M \wedge m_1 \neq m_2 \wedge V_{u_1,m_1} \in V_{u_1} \wedge V_{u_2,m_2} \in V_{u_2} \wedge V_{u_1}, V_{u_2} \in V \wedge x \in V_{u_1,m_1} \rightarrow x \in V_{u_2,m_2}$ **then**
    if $s_p \neq personal$ **then**
      $s_p \leftarrow global$
    end
  else
    $s_p \leftarrow local$
  end
  $t \leftarrow t_{+1}$
until $t \neq t_{+1}$

UoLs in the time interval. If it is, the scope of the property is set to *global*, and to *local* if it is not.

The second pillar of IMS LD adaptation mechanism is a set of rules. Rules are structured as a conditional *if-then-else* control flow and therefore evaluate the *then* part when the *if* condition is verified and the *else* part when it is not. The operators used in the *if*, *then*, and *else* blocks are defined in the IMS LD

Level B of the information model [3] and include classic logical, arithmetic, and comparison operators, and others more specific that support operating on the user model and the process structure (i.e., methods, acts, role-parts, and activities). Contrary to programming languages, the semantics of the *then* part is limited and so can only contain actions, which will show or hide objects, change a property value, or notify a role. The *else* part is more standard, and allows concatenating *if-then-else* control structures, in addition to actions.

Notice that IMS LD rules structure is very similar to those described in Section 4. Two additional remarks about the transformation of the decision tree. Firstly, there is always a viable transformation between the obtained decision tree and the IMS LD rules since *(i)* the operators used by the decision tree are supported by IMS LD and *(ii)* we just learn two types of actions for changing *(i)* the visibility of an object and *(ii)* the value of a property. Secondly, the rules we generate have the structure of an *if-then* control flow, that is, we never have an *else* block. This last feature does not limit the semantics of our rule model since each *else* is represented as an *if-then* rule but with its specific condition.

Finally, we must also mention that in IMS LD rules are evaluated at the beginning of a session and each time a property is changed. Furthermore, rules are also evaluated at the completion of methods, plays, acts, and activities. This is reasonable since these control structures coordinate the participants of the UoL and thus that or teachers change the value of their properties in the activities, methods, plays, and acts define a group behaviour and thus changes must affect all the members of the group. However, this behaviour is very dependent of IMS LD and even more complicated to extract from the events of log files. In fact, usually this kind of adaptation is represented by the process mining as a simple activity that changes the state of the properties. Consequently, the completion of upper hierarchical structures, such as plays and acts, will not change the state of learning. However, this is not a big issue since our solution defines specific dummy activities to such a change.

6 Conclusions and Future Work

In this paper we have proposed a global approach that facilitates the reuse of UoL 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. From these two results, we also describe how to translate these models into an IMS LD UoL.

Finally, as future work we plan to extend the number of operators used by the decision tree algorithm to so give support to the complete grammar or IMS LD.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st edn. Springer Publishing Company, Incorporated (2011)
2. Bergenthum, R., Desel, J., Harrer, A., Mauser, S.: Modeling and Mining of Learnflows. In: Jensen, K., Donatelli, S., Kleijn, J. (eds.) ToPNoC V. LNCS, vol. 6900, pp. 22–50. Springer, Heidelberg (2012), http://dl.acm.org/citation.cfm?id=2231056.2231058
3. Global Learning Consortium, I.: IMS Learning Design Information Model, Version 1.0 Final Specification (March 2003), http://www.imsglobal.org/learningdesign/index.html
4. Dalziel, J.: 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), pp. 51–58. IADIS Press, Adelaide (2003)
5. Derntl, M., Neumann, S., Griffiths, D., Oberhuemer, P.: The conceptual structure of ims learning design does not impede its use for authoring. IEEE Transactions on Learning Technologies 5(1), 74–86 (First 2012)
6. Hernández-leo, D., Villasclaras-Fernández, E.D., Asensio-Pérez, J.I., Dimitriadis, Y., Jorrín-Abellán, I.M., Ruiz-Requies, I., Rubia-Avi, B.: Collage: A collaborative learning design editor based on patterns. Educational Technology & Society 1(9), 58–71 (2006)
7. Karampiperis, P., Sampson, D.: A flexible authoring tool supporting adaptive learning activities. In: Proc. of IADIS International Conference on Cognition and Exploratory Learning in Digital Age (CELDA 2004), pp. 51–58. IADIS Press, Lisbon (2004)
8. Martinez-Ortiz, I., Sierra, J.L., Fernandez-Manjon, B.: Authoring and reengineering of ims learning design units of learning. IEEE Trans. Learn. Technol. 2(3), 189–202 (2009), http://dx.doi.org/10.1109/TLT.2009.14
9. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
10. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)
11. Reimann, P., Yacef, K.: Using Process Mining for Understanding Learning. In: Handbook of Design in Educational Technology, pp. 472–481. Routledge, New York (2013)
12. Vidal, J.C., Lama, M., BugaríN, A.: Petri net-based engine for adaptive learning. Expert Syst. Appl. 39(17), 12799–12813 (2012), http://dx.doi.org/10.1016/j.eswa.2012.05.013
13. Weijters, A., van der Aalst, W., Alves de Medeiros, A.: Process mining with the heuristics miner-algorithm. BETA Working Paper Series WP 166. Eindhoven University of Technology (2006)
14. Wen, L., Aalst, W.M., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. 15(2), 145–180 (2007), http://dx.doi.org/10.1007/s10618-007-0065-y