MODELLING A LAYER FOR REAL-TIME MANAGEMENT OF INTERACTIONS IN WEB BASED DISTANCE LEARNING

CARLOS SOUSA PINTO[1]; FERNANDO M. S. RAMOS[2]

[1]Department of Information Sistems, School of Engineering, University of Minho
Campus de Azurém, Guimarães, Portugal
e-mail: csp@dsi.uminho.pt
[2]Department of Communication and Art, University of Aveiro
Campus de Santiago, Aveiro, Portugal
e-mail: fmr@ca.ua.pt

Abstract

In the last few years, the University of Aveiro, Portugal, has been offering several distance learning courses over the Web, using e-learning platforms.
Experience showed that different editions of a same course, using the same contents and structure, and having similar target learners, had different success rates. What would be the reason for that?
A hypothesis was considered: The level of success could be directly related with the remote follow-up of the learners' participation in the courses; the best results usually occur when the follow-up is closer.
The existing e-learning platforms offer and the standardization works being developed by organizations and consortiums like IMS (IMS Global Learning Consortium, Inc), ADL SCORM (Advanced Distributed Learning Sherable Content Object Reference Model), IEEE LTSC LOM (Institute of Electrical and Electronic Engineers Learning Technologies Standard Committee Learning Object Metadata), ARIADNE (ARIADNE Foundation for the European Knowledge Pool), AICC CMI (Aviation Industry CBT Committee Computer Managed Instruction), etc, don't cover the course monitorization concerns mentioned. Those projects were focused on aspects like contents and its delivery in the context of the execution of the courses' activities. This is even true in the SCORM project that doesn't include any reference to the management of the e-learning processes.
Recently, in the context of the IMS Global Consortium, a new project designated IMS LD (Learning Design) is under development, providing a framework for the description of learning units under a three level model. In the most recently defined level, the C level, some functionalities related to notifications were proposed, expressing similar concerns to the ones that triggered our research. However, the extent at which IMS LD takes the functionalities is, from our point of view, not complete.
This article describes a proposal of a reference model and functionalities towards a specification of a layer for real-time management of user interactions on LMSs, and its possible integration with the ADL SCORM standard proposal. The paper includes a discussion of the management metadata model for the LMS sub-system and how the integration of the management module under SCORM may be achieved.

Keywords: e-learning; meta-data; standards, SCORM, IMS, Web based learning

1. INTRODUCTION

Thinking on Web based courses, we can identify activities to be executed not only by learners but also by teachers and elements of support teams. The success of those teaching/learning experiences depends on the right participation of each of those types of actors.

Sometimes, for a learner or for all the participants in a course, the objectives established for that course are not reached, and we can identify several reasons for that. For example, if someone does not execute a programmed activity, inside the defined timing, it can compromise the rest of the course to that person.

Let's see another example involving the teacher and the learners. Let us assume that an activity A2 is programmed to be executed by the learners and that it depends on the previous knowledge of the result of the evaluation of a work submitted by the learners to the teacher (activity A1). If the teacher doesn't inform the learners about that classification in useful time, that can compromise the execution of the activity A2. In such situations it would be important to detect the fault and to take some corrective actions. Unfortunately, many times, the identification of the problems that leads to the insuccess occurs too late, compromising any type of solution for those problems.

We belive that LMSs should include mechanisms for automatic monitorization of the participations, so that the probability of success of the teaching/learning process could be enhanced.

2. OUR CONCEPTUAL MODEL

Our proposal for the management layer lies in the monitorization of an informational entity that we call "events" and in its comparison with another one that we assign as "activities". This last one implements the structure of the course while the first reflects the interactions of the actors with the LMS, in what concerns the execution of the scheduled activities.

The subsystem of management completes itself with the inclusion of a component of notifications and with the definition of a set of rules that regulate the notification process.

This functionality foresees the existence of three different instants where notifications can occur:

- Before the beginning of the activity (Warning);
- Before being reached the limit defined for the execution of the activity (First alarm);
- After this limit has been exceeded (Second alarm).

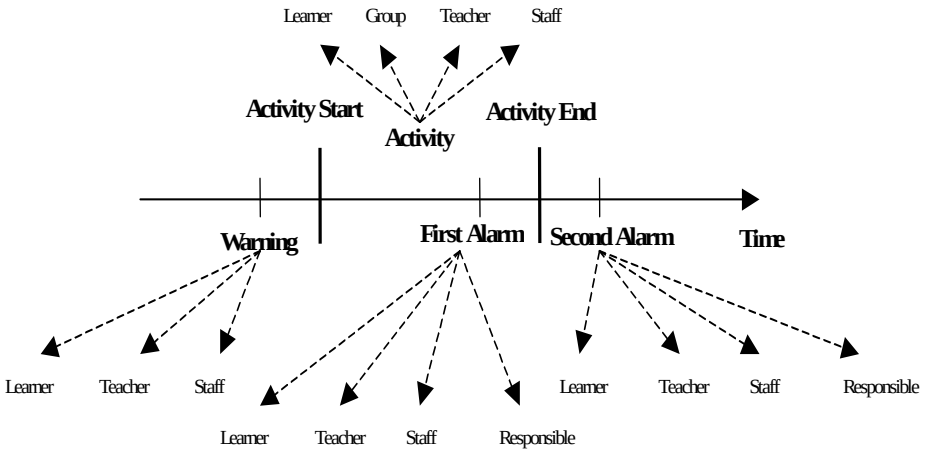

FIGURE 1 - ATOMIC UNIT OF MANAGEMENT IN REAL TIME OF ACTIVITIES

Figure 1 represents what we call "Atomic Unit of Management in Real Time of Activities", on the basis of which all the courses can be architected.

For us, a course can be any combination of units of this type, organized in a sequential, parallel or random way and including the possibility of recursive application of this concept to the decomposition of an activity in subactivities, to be executed by an actor or a group of actors.

As we can see in the Figure 1, our model includes several types of possible actores. In the documentation about the principal projects on this subject [3], [4], [5] only learners are referred and we can't read anything about the participation of groups, teachers and members of support teams.

In the same way, we can't see any references to group activities. That is, activities composed by sub-activities as showed in Figure 2, each of them to be executed by an element of the group.

Figure 2 shows an activity composed by sub-activities, each of them having exactly the same set of proprieties referred above. In that figure we represent the sub-activities as sequencial but it was possible to include sub-activities to be executed in a parallel way.

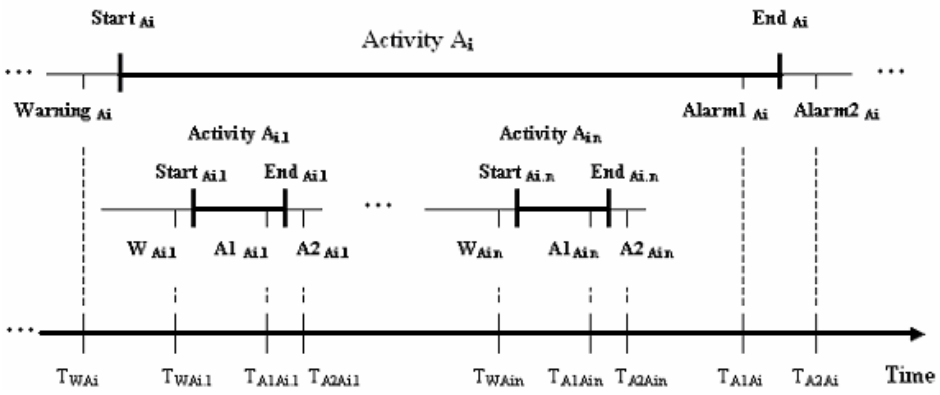

FIGURE 2 - ACTIVITY COMPOSED BY SUBACTIVITIES

In accordance with Figure 1, each activity has a "warning" emitted before the instant defined for the beginning of the activity, to alert the actors to the proximity of that activity. This type of notification makes sense only if the activity is not a random one. In these cases the activity is initiated by the choice of the actor and not by the occurrence of a defined trigger.

When the activity is initiated, its conclusion must occur inside the defined window of time.

Before reaching the deadline to the execution of the activity it must be tested if it was already terminated or if it is still running. If this is not the case, a "first alarm" will be generated.

In this way it can be prevented that the structure of the course has to be redefined and the system will potentially contribute for the increase of the probability of success of actors' participation in the course.

Finally, once it is possible that an actor misses the execution of an activity inside the foreseen window of time, the system will have to emit a third type of notification, a "second alarm", destined to make possible the adoption of corrective actions, namely the reprogramming of the activity for this actor.

Figure 1 shows that a course can include activities to be executed by learners, groups, teachers and elements of the team of support (staff).

We can also see that notificatios can be sent to different destinations, depending on the type of notification (warning, first alarm or second alarm).

3. INTEGRATING OUR WORK INTO ADL SCORM

Once there are several organizations and consortiuns, involving the industry, governmental institutions and the universities developing works of standardization, it seemed important to see how the referred management aspects were covered by these works, and to perceive how it could be possible to articulate our work with the ones available from these organizations and consortiuns. Given the existence of the already referred works of standardization (IMS, AICC, ARIADNE, ADL, IEEE) and once the project ADL SCORM is the one that congregates greater number of contributions from other projects, we thought that it would be interesting to develop our work towards its possible integration in the SCORM. Being so, we made the identification of potential points of interface between our management layer and other layers referred in the SCORM 2004 specification [3], to allow the monitorization of the interactions with the LMSs (Learning Management Systems). This work leads to the identification of SCORM behaviors and elements of metadata that need to be enhanced and to the definition and inclusion of procedures in our subsystem of management, capable to make compatible this new layer with the functionalities that already exist in the SCORM project.

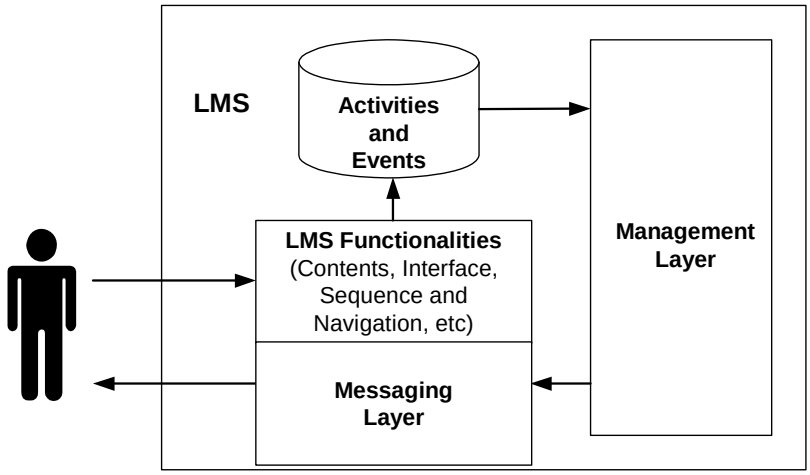

FIGURE 3 - ARCHITECTURE OF MANAGEMENT LAYER RELATIONSHIP WITH OTHER LMS COMPONENTS

Figure 3 represents our perspective of the integration of the proposed management layer and the different modules of an LMS and the way they should related to each other.

We should read the scheme of Figure 2 as follows:

1. The authors interact with the platform in order to construct the courses, registering among other information, the one that implements the structure of the course itself, that is, the activities.
2. Later, the actors, in order to execute their activities, will interact with the LMS and, during this interaction, the LMS promotes the register of the diverse corresponding “events".
3. The actors will be able to use the mechanisms of synchronous and/or assynchronous communication, to communicate informally between them.
4. Permanently, the layer of management of the LMS will consult the repository of activities and events to identify situations that justify the emission of notifications. If there are cases

that justify this emission, the management layer will request the "messaging" layer of the LMS, passing to it, pairs with the following constitution:

- Identification of the destination;
- Message

5. Finally, the LMS using its functionalities of messaging", after identifying the preferential way of communication of each destination, will send the notifications, according to the information received from the management layer, or it will create the conditions so that these notifications are sent in a non electronic form.

It should be highlighted that we can have more than one destination for a notification, namely when sending messages for a group of learners, for example. Even the case of destinations of different types, eventually receiving different messages, is well supported by our management layer as it can be inferred from the structure of informational pairs showed above in point 4.

In order to integrate our proposed management layer with the LMSs builded under SCORM recomendations, it is necessary that the LMSs can create the information about the execution of the activities in our informational entity “events”. The registration of that information must be done only if the activities are terminated successfuly. In our point of view, an activity for which there is no “event” registration, is an activity not executed and the management layer must generate notifications related to that fact.

The integration of our work with SCORM proposed guidelines can be done at several levels. In the SCORM RTE (Run-Time Environment) documentation, we can read that during the execution of a SCO (Sharable Content Object), that was launched by the LMS, the SCO finds an instance of the API (Application Programming Interface) and iniciates the communication between itself and the LMS by calling the methods pertained to the API. Those methods are distributed by three main groups – Session Methods, Data-transfer Methods and Support Methods.

The session methods – “Initialize()” and “Terminate()” - are used to initiate and terminate the communication while the data-transfer methods – “GetValue()”, “SetValue()” and “Commit()” – are used to manage the storage and retrieval of data to be used in the actual communication session [3]. The method “SetValue()” is used to send information from SCO to LMS, for storage.

We think that it is possible to extend the behavior of this component of the LMS’ API so that it could promote the insertion of right information in our “events” informational entity, in the cases that it is required.

Accordingly to SCORM documentation [7], LMSs must use SCO reported information, so that it could be possible to take decisions about the sequence of the next activities to be delivered. If the SCO, using the SCORM RTE Data Model element “cmi.completion_status”, informs that the learner has completed that SCO, the activity to which that SCO belongs must be considered terminated too. So, the LMS can extend this mechanism to create a valid entry in our proposed informational entity “events”. On the other hand, the data model element “cmi.time_limit_action”, indicates what the SCO should do, when “cmi.time_limite_action” is exceeded [7].

In our work, we also have considered an instant, after the defined end, to test if the activity was executed and to decide what to do next.

We can identify a third possibility of integration of our work with SCORM proposals. In the SCORM Sequencing Behaviour Pseudo Code [7], we can read that the attribute “Objective Satisfied Status” must be set to true when an objective is reached. It is also a good time to potentially create an instance of our informational entity “events”.

4. THE META-DATA MODEL

Figure 4 is the hierarchic meta-data model corresponding to our vision of a course. In that model, we represent more than the elements strictly related to the problem of management we are discussing in this paper.

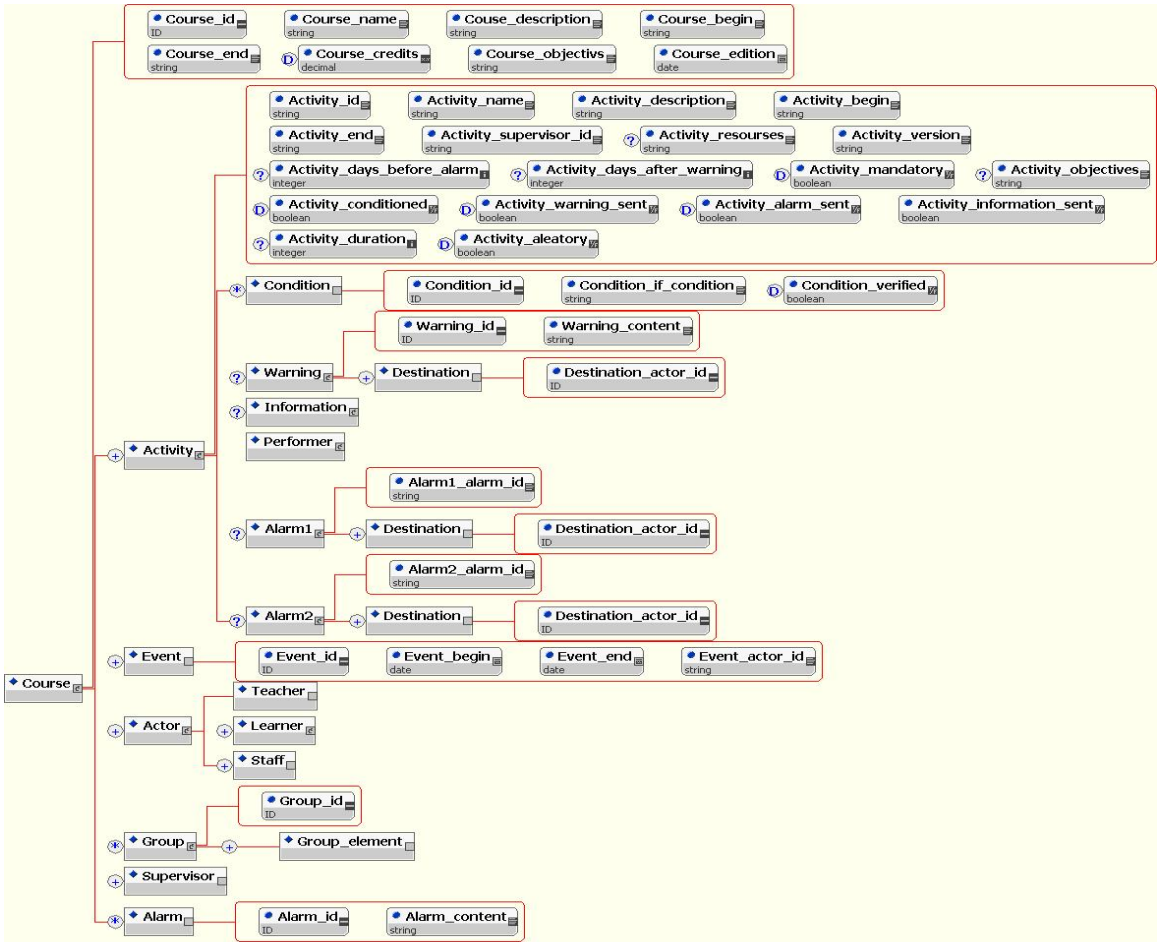

FIGURE 4 - PARCIAL VIEW OF HIERARCHY OF META-DATA ELEMENTS

In fact, the model represented in Figure 4 is a data model that could support a complete LMS, accordingly to our perspective of what an LMS should be.

The symbols before the elements and attributes have the following meaning:

"+" - The element can exist one or more times.
"*" - The element can exist zero or more times.
"?" - The element is opcional.
"D" - The attribute has a default value.

All the elements and attributes without any precedent symbol are mandatory and must exist one time.

In the model of the Figure 4 we have included the elements “alarm1” and “alarm2” without a “content”attribute because there are no conceptual differences between the two types of alarms. Only the timing of eventual appearance in the process is different. This way, the two elements have an attribute (Alarm1_alarm_id and Alarm2_alarm_id) that points to the element “alarm” where all the possible alarms must be stored.

It is clear in the meta-data model that an alarm (first alarm or second alarm) can have more than one destination, as referred above.

Table 1 represents an excerpt of the metadata table where we define all the elements and atributes of the model showed in Figure 4.

TABLE 1 – Excerpt of meta-data table for the element “activity”

Attribute name	Documentation	Type	Allowed Values	Cardinality	Default
activity_id	This attribute must be filled with a valid activity’s identifier.	String		1:1	
activity_begin	This attribute must be filled with the beginning date and time of the activity (the date and time verified at the beginning of the execution, for aleatory activities).	String		1:1	
activity_end	This attribute must be filled with the date and time of the end of the activity (the sum of *activity_duration* to *activity_begin*, for aleatory activities).	String		1:1	
activity_days_after_alarm1	This attribute must be filled with the number of days of antecedence for delivering a first alarm (zero, if there is no first alarm).	Integer		0:1	
activity_days_before_alarm2	This attribute must be filled with the number of days between the end of the activity and the delivery of the second alarm (zero, if there is no second alarm or if it must be delivered immediately after *activity_end*).	Integer		0:1	
activity_days_after_warning	This attribute must be filled with the number of days between the delivery of the warning and the beginning of the activity (zero for aleatory activities).	Integer		0:1	
activity_warning_id	This attribute must be filled with a valid identifier that	String		0:1	

	must be unique (without content for aleatory activities).				
activity_alarm1_id	This attribute must be filled with a valid identifier that must be unique (without content for activities without first alarm).	String		0:1	
activity_alarm2_id	This attribute must be filled with a valid identifier that must be unique (without content for activities without second alarm).	String		0:1	
activity_father_id	This attribute must be filled with the value of *activity_id* or with the *activity_id* of the activity it derives from, if it is a sub-activity of another one.	String		1:1	
activity_performer	This attribute must be filled with valid identifiers of the actors that must execute the activity.	String	Actor_id or group_id	1:*	
activity_performer_type	This attribute must be filled with one of the allowed values.	Symbol	All, All_learners, Group, Staff, Teacher, Learner	0:1	
activity_duration	This attribute must be filled with the number of days allowed for the execution of the activity.	Integer		1:1	

Observing table 1, we can see the attribute *activity_performer_type* and its allowed values. Analysing the content of this attribute it will be very simple for the LMS to identify all the actors that must perform the activity. For example, if the LMS encounters the value “All_learners”, it must select all the actors of the type learner participating on the course and the attribute *activity_performer* must be ignored. The same mechanism must be used if the content of the attribute is “All”. In this case, all the actors participating on the course must be selected as performers of the activity. However, if the value encountered is “Group”, the LMS must read the attribute *activity_performer* in order to identify all the groups which must execute the activity. For each identifyer of a group the LMS must retrieve its members in the element “Group” (see figure 4).

Finally, if the content of the attribute *activity_performer_type* is “Staff”, “Teacher” or “Learner”, the attribute *activity_performer* will contain the identifyer of the actor that must execute the activity.

We would like to highlight the fact that for us the elements of the support team (staff) must be considered as a single entity. This way an activity to be executed by “Staff” should exist only one time in the course. We have the same simplification in the mechanism of notification. For us it is irrelevant what member of the support team must execute an activity or should be notified in a particular situation.

We also consider activities to be executed by a single learner. In the context of Web based learning it is not normal to plan an activity to a particular learner. However, we can have some situations that justify such a practice. It is the case that a learner must execute an extra activity in order to recover from an abnormal performance in wich he has failed the expected sequence of the foreseen activities.

TABLE 2 – Excerpt of meta-data table for the element "event"

Slot name	Documentation	Type	Allowed Values	Cardinality	Default
event_id	This slot must be filled with one of the activity identifiers.	String		1:1	
event_begin	This slot must be filled by the LMS with the date and hour of the beginning of the execution of the activity.	String		1:1	
event_end	This slot must be filled by the LMS with the date and hour of the end of the execution of the activity.	String		1:1	
event_actor_id	This slot must be filled with with one of the identifiers registered in the actor.	String		1:1	

Seeing table 1 and table 2, it is easy to understand how the management layer we propose works. Almost all the relevant information exists in the element "activity" at the beginning of the course. Only some information related to any aleatory activity is to be provided after that instant.

During the execution of the course the LMS must generate instances of the element "event", one for each planned activity and for each actor. This generation and storage must occur only when an actor finishes an activity with success.

We would like to remember that it is not a responsibility of the management layer to decide if an activity is completed with success or not. The proposed management layer only takes the information stored into the two informational entities (activities and events) and compares that information to decide if it is necessary to notify somebody.

5. CONCLUSIONS

The standardization works being developed are very important because they will allow the uniformization of the LMSs and contents development. This is a key aspect in order to obtain greater levels of reuse and interoperability among different systems. However, it is clear that those works have as principal concerns, the contents development, the schedulling of the activities to be executed inside the courses and mechanisms for sequencing and navigation over the contents.

Aspects that we consider important, like real-time monitorization of the different actors participation, are not considered.

Our experience in Web based distance learning indicates that when there are not an efective follow up of the activities by the responsible for the courses, the probability of insuccess grows up.

On the other hand, it seems to be an incomplete approach to consider only learners as actors participating in a course and that is what we can see in the different projects documentation, namely in the SCORM and IMS documentation.

Teachers and members of the support teams are also important actors to be considered in the execution of the courses and it is very easy to identify several activities to be executed by them.

Based on these considerations we have developed the work presented in this paper, having in mind the proposal of a reference model and functionalities towards a specification of a layer for real-time management of user interactions on LMSs, and its possible integration with the ADL SCORM standard.

Our proposed management layer can detect deviations to the course scheduled activities, enabling the correction of these deviations in useful time. This is possible due to a component of automatic notifications that is also responsible for the detection of abnormal situations

The validation of the work is not complete at this time. It is necessary to effectively integrate our management layer in a SCORM complient LMS and to use this e-learning platform in a significant number of experiences of distance learning. After these experiences it will be possible to compare the results with those known from passed experiences.

REFERENCES

[1] Ramos, F. Moreira and Santos, A., "Towards a Reference Framework for eLearning Management", Proceedings of DLA'2001 - Distance Learning Administration Conference, University of West Georgia, GA, USA, (2001), June 6-8.

[2] Ramos, F., Moreira, A., Santos, A., Conde, A., Neves, L., Peixinho, F., Pinto, C. S., "Tracing Dynamic Behaviours in Distributed Learning Communities", Proceedings of 16th Australian International Education Conference, Hobart, Tasmania, Australia, (2002), September 30 – October 4.

[3] ADL, Advanced Distributed Learning, "SCORM (2004), ADL Sharable Content Object Reference Model, Version 1.3.", http://www.adlnet.org, (2004).

[4] IMS Global Learning Consortium, Inc., "IMS Learning Design Information Model", http://www.imsglobal.org/learningdesign/ldv1p0/imsld_infov1p0.html, (2003), January 20.

[5] IEEE LTSC, Learning Technology Standards Commitee, "Draft Guidelines for Learning Technology", http://ltsc.ieee.org, (2001), January 30.

[6] IEEE LTSC, "P1484.11.1 Draft Standard for Learning Technology - Data Model for Content Object Communication.", http://ltsc.ieee.org, (2003), November 10.

[7] ADL, Advanced Distributed Learning, "SCORM Run-Time Environment Version 1.3.", http://www.adlnet.org, (2004).