Simplified Workflow Representation of IMS Learning Design*

Juan C. Vidal, Manuel Lama, and Alberto Bugarín

Department of Electronics and Computer Science
University of Santiago de Compostela, Spain
{juan.vidal,manuel.lama,alberto.bugarin.diz}@usc.es

Abstract. A way of describing the teaching, learning interactions and activities is through Educational Modeling Languages. These languages are formalized so an interpreter can automatically coordinate the activities. However, like for example IMS LD, these languages are usually difficult to understand causing that instructors do not always understand the way the designed unit of learning is coordinated. In this paper we present a graphical notation that simplifies the authoring process of new and preexisting units of learning specified in IMS Learning Design. The graphical notation extends the YAWL language and approach the design of units of learning through a set of patterns distributed in three hierarchical layer.

Keywords: Adaptive Learning, Visual Language, Learning Design, Workflows, Petri nets.

1 Introduction

In recent years, a significant effort has been invested in the definition of Educational Modelling Languages (EMLs) to facilitate the design of a course *from a pedagogic point of view*. One of the results was the IMS Learning Design (IMS LD) specification [1] that has emerged as the *de facto* standard for the representation of learning designs. From the authoring point of view, IMS LD is structurally complex and designers have difficulties to create and understand their units of learning. To deal with this issue an important effort has been devoted to the definition of visual languages that facilitate instructors the design of their units of learning (UoL) [2,3,4,5,6].

From these approaches those that represent the learning flow of IMS LD like a workflow are particularly interesting. These approaches usually coordinate the learning activities by means of a set of workflow-based control structures [7,8,9] with the aim of abstracting the edition of an IMS LD unit of learning. However, these approaches fail in not considering the learning flow of the IMS LD

* Authors wish to thank the Xunta de Galicia and the Ministerio de Educación y Ciencia for their financial support under the projects 09SIN065E and TSI2007-65677-C02-02.

© Springer-Verlag Berlin Heidelberg 2011

specification like a hierarchical workflow, which is composed of a set of layers: the first layer is related to the method structure (that is, the plays); the second layer defines the acts that represent each of the plays; and the last layer models the coordination of the learning activities. With this approach, the one we followed in this paper, the creation of IMS LD units of learning consist in building a complex model from simple workflows, facilitating to users a complete view of the learning flow of a UoL.

In this paper we present a graphical user interface that provides an alternative to the design of UoLs that follow the IMS LD specification. In this GUI we address the design of UoLs through a visual model that has two objectives. On the one hand, we pretend to simplify the creation of UoLs through a set of visual patterns with a limited interaction with the user. Visual languages provide an important advantage over text-based: they are more legible. For example, it is much easier to understand a course described by a visual language than a course specified in the XML format of IMS LD.

On the other hand, we want to give to instructors a complete view of the structural characteristics of the course they are defining. Thus, the main workspace of the GUI shows the whole view of the IMS LD learning flow of a UoL. One of the main problems of IMS LD is its difficult interpretation: instructors must be experts in IMS LD to design a course. This is due to the complex structure (based on the theater metaphor) that IMS LD implements to coordinate students with the activities they must perform, which is not always well understood by instructors. For this reason, a visual representation is a major help to understand how the course is from a structural point of view.

Taking into account that a UoL can be viewed as a flow of learning where students and instructors coordinate the activities they have to perform, in [10] we presented OPENET4LD, a Petri nets-based player for UoLs specified in IMS LD. Petri nets are one of the most popular languages for modelling workflows [11,12] because they are a mathematical formalism and, furthermore, they have a graphical representation. They are also a very expressive language that can represent most of the workflow patterns that exist today [13]. However, as seen in [10], models constructed by these networks can become large and quite complex to understand. Since one of our goals is to facilitate the design of UoLs by people without experience in the modelling of workflows, we decided to use a language with a simpler visual syntax than the one provided by Petri nets. The language chosen was YAWL (Yet Another Workflow Language) [14], an extension of the Petri nets language, which provides a bunch of new connectors that represent the typical control structures of workflows. For example, it provides specific structures for selecting, splitting, synchronizing or merging of the execution thread. In a certain sense, the relationship of YAWL with Petri nets is the same as the one defined between third-generation programming languages with assembly languages, that is, they provide a higher level of abstraction that hides complexity to the designer.

Based on the syntax defined by YAWL, in this article we will describe the visual models we use to represent IMS LD and its mappings with the models of Petri nets defined in OPENET4LD, which ultimately will confer them their operational semantics. Finally, we will describe the implementation of the GUI and how is its integration with OPENET4LD.

2 Visual Model for the Representation of Units of Learning

In this section we detail the visual models that simplifies the way in which instructors design their UoLs based on the IMS LD specification. These models or patterns are described using the visual syntax of YAWL, combined with the properties of hierarchical nets, to so facilitate their representation. Unlike Petri net-based models of OPENET4LD, that represent the complete execution of the UoL, the models described in this section are focused only on the structural properties of IMS LD. In this sense, the simpler the visual model, the easier to use. Therefore, our visual models will not control how to stop, suspend or resume the methods, plays, acts or activities of IMS LD. These details will be transparent to users and, as we will discuss in Section 3, will be provided automatically by OPENET4LD.

The model we present in this section is structured in a hierarchy of three layers. The first layer deals with methods modeling, the second with plays, while the third applies to the execution entities, that is, to activities, structure of activities, and units of learning. For each of these layers, we capture the operational semantics of the element, such as it is defined by IMS LD, and transform it into a YAWL model.

Finally, we must remark that in this article we only will describe the features of our models for the level A of IMS LD. In this level, IMS LD defines methods, plays, acts and activities, and the structural properties of these elements, which is the purpose of this article. However, we also plan to offer level B support through the GUI.

2.1 Representation of Methods

A method implies the execution of a set of plays, although some of these plays are optional, and IMS LD does not impose any requirement on the way these plays can be executed. Figure 1(a) introduces the pattern we use to capture the control that the method establishes over its plays. In this example, the figure shows the execution of three plays, although this model can be extended to represent the parallel execution of n plays. We must emphasize several components of this figure:

– A start place of the method. The circle with a triangle represents the point where the method execution begins.
– A point where the execution of all the plays are parallelized. This point is represented by the AND-SPLIT pattern of YAWL. However, the semantics of this pattern is little different since the execution of the parallel structure will not synchronize until the set of mandatory plays have been executed.

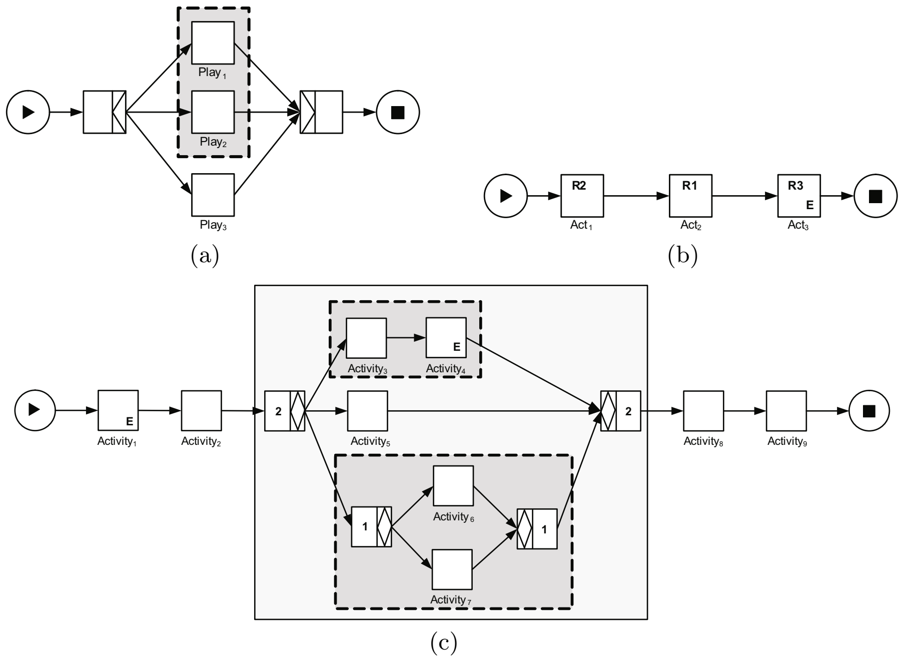


Fig. 1. Example of the patterns for IMS LD representation. The pattern (a) controls the parallel execution of plays, (b) the sequence of acts and (c) the execution of a role part of an act.

- In the central part of the figure, rectangles represent the set plays to run in parallel. The rectangles represent tasks in YAWL, which can be simple or complex. In the case of this model, each play is a complex task that have an internal structure (hidden in the model) defined by the acts of the play.
- A point where the execution of the plays is synchronized. This point is represented by the AND-JOIN pattern of YAWL.
- An end place of the method. The circle with a square represents the point where the method finishes executing.

In order to facilitate the understanding of the model, we extended the visual syntax of YAWL with a gray colored rectangle that identifies the plays of the method whose execution is mandatory.

2.2 Representation of Plays

A play consists on an ordered execution of a set of acts. YAWL does not have a specific pattern to represent sequences. In YAWL a sequence is established based on the dependencies between the elements, where dependencies are determined by the directed arcs between the visual elements. For example, Fig. 1(b) depicts this structure with a sequence of three acts, where Act_1 is the first task to be performed and Act_3 is the last one.

As in the case of plays in the method model, acts are represented by means of rectangles. These rectangles also represent complex tasks, although in this case their internal behavior will coordinate the execution of the role-parts of the act. Therefore, acts will be associated with a set of nets representing each of the role-parts to be made, unlike the case of plays that were associated with a unique net.

In addition we have extended the notation defined by YAWL to enable designers to see if an act has a role-part: their shape is annotated with a R accompanied by a number. For example, in Fig. 1(b) the Act_1 act is annotated with $R2$ which means that it is associated with two role-parts. In the case of acts 2 and 3, the notation indicates that they have one and three role-parts, respectively. The same applies with the environments associated with acts, although in this case they are denoted by an E. Therefore, when an act has any environment its shape is annotated with a E at the bottom left.

2.3 Representation of Role-Parts

The representation of a role-part is more complicated. In the case of methods and plays the structure of the net is stable and varies only when a new element is added to the model. For example, a new play in a method is just a new parallel branch within the AND-split and AND-JOIN patterns. By contrast, the design of a role-part does not have a fixed structure, its structure is defined by combining:

– *Sequence of activities.* They are defined with a sequence structure, as we have described in Section 2.2, but with the restriction that only simple activities, selection of activities, and UoLs can be part of this sequence.
– *Selection of activities.* They are defined within a structure formed by an OR-SPLIT and an OR-JOIN, although in our model both constructs take a sightly different meaning: the execution thread remains in this structure until a number of execution entities (selections, sequences, simple activities, or UoLs) have been selected. The number of execution entities that must be selected is represented as a number in the center of the OR-SPLIT/OR-JOIN control construct.
– *Simple activities.* They are the leaves of our hierarchical model and do not break down into simpler elements.
– *Units of learning.* They are represented as activities, through a rectangle, but represent a complex task. In this case, the task will execute a method model as we have described structure in Section 2.1.

An example of role-part is shown in Fig. 1(c) which represents a sequence of five activities, where the third one is a selection of activities. This figure also shows how different elements are combined, in this case one of the selection of activities is made by a sequence of activities, a simple activity and another selection of activities. As in the case of acts, environments may be associated with activities, being denoted by an E in the bottom left of the rectangle.

3 Hiding the Complexity of Our Visual Model

The visual model described in the previous section only shows part of the operational semantics that supports the execution of UoLs. Our visual model covers only structural features of UoLs. For example, it only contemplates the coordination of plays in a method, of the acts in a play, or of the role-parts in an act. If we want to have a complete model it is necessary to complement the visual model with other features more closely linked to the management of the execution.

In this paper we complemented our model with the approach followed in OPENET4LD. Behind OPENET4LD there is a set of Petri nets that support the implementation of UoLs and also provide a set of features that facilitates the management of methods, plays, events and activities. As a result, these items can be suspended, resumed, or stopped, and this action will be propagated to items that are hierarchically dependent on him.

3.1 Petri Nets-Based Workflows of OPENET4LD

OPENET4LD is a UoL player that allows uploading UoLs stored in the XML format defined by the IMS LD specification. The main difference with other IMS LD engines currently available, such as CopperCore [15], GRAIL [16], MOT+ [17] or LAMS [18], is that OPENET4LD is based on a workflow formalism, specifically on hierarchical Petri nets, so it may verify the structural properties and the correction of the implemented courses.

The transition from IMS LD is complicated and generates structurally complex Petri nets. This is because IMS LD introduce control elements that restrict the order in which activities can be executed. This control structure is based on the theater metaphor where each of the items (plays, acts, activities) requires a particular coordination. OPENET4LD implements two types of Petri nets associated with each IMS LD element. The first net was designed to unify the way in which a method, play, act, or activity is executed independently of the level in which it is situated in the hierarchy. This net is represented in Fig. 2 and is used to indicate the current state of execution of the element. As it can be seen in this figure, the net consists of four parts. The first part is focused on the transition *run-element*, and is responsible of the execution of the element. The second and third part control the suspension and resumption of the element triggered by the user or by a hierarchically higher element. Finally, the last part controls the stopping of elements motivated by the stopping of a hierarchically higher element, by a user trigger, by a time limit consumption, or by a certain condition established in IMS LD.

The second net captures the control structure for the execution of methods, plays, acts, and structure of activities. Each of these structures reflects the operational semantics defined by IMS LD and transforms it into a Petri net. For example in the case of a method, this transition is replaced by a net with a set of parallel transition representing the mandatory (defined in the when-play-completed IMS LD property) and non-mandatory plays of the method.

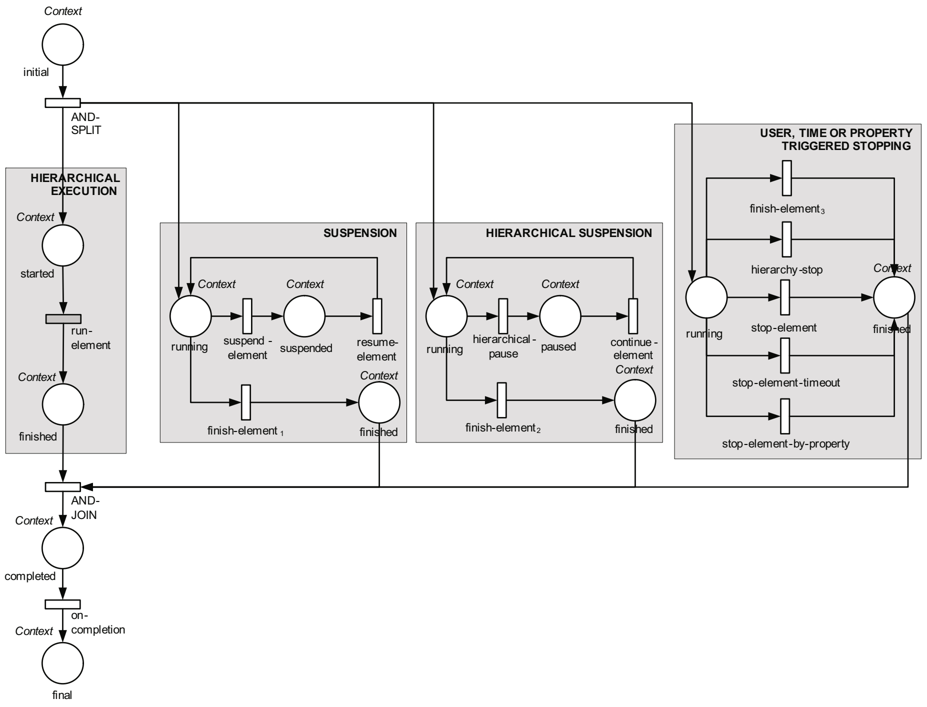


Fig. 2. Petri net that controls the execution, suspension, resumption and completion of methods, plays, acts, and activities of the unit of learning

In the OPENET4LD model, an element (such as method, play, act or activity) is therefore composed of a control net and a structure net. The union between the two nets is done through two composition mechanisms of hierarchical Petri nets: fusions and substitutions. For example, the union between a control net and its structure is done by substituting the transition *run-element* of the control net by a Petri net with the appropriate control structure. In the case of a method, this transition is replaced by a net with a similar structure as the Petri net depicted in Fig. 3. This substitution allows methods to coordinate the execution of their plays. It should be mentioned that this process is repeated for each hierarchical level of IMS LD, that is, for each method, play, and act. The union between the various components is also carried out through transitions substitution. For example, transitions *run-play$_i$* of Fig. 3 by a control net integrates the execution of the plays with the Petri nets that represent the method execution. This feature gives to our hierarchical model a tree-shaped structure where each tree level alternates a control net with a structure net, and where tree leafs are the activities to perform.

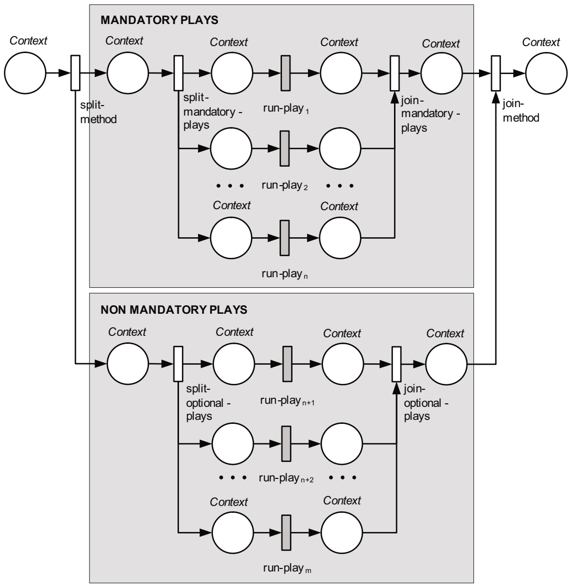


Fig. 3. Petri net that controls the execution of a set of plays

3.2 Mappings between OPENET4LD and the Visual Model

We have established a set of mappings between our visual models and the Petri nets models of OPENET4LD. Through these correspondences, our visual model is transformed into an executable model based on Petri nets. This combination is an important advantage, because (i) it takes advantage of the simplified approach for designing UoLs that our visual models provide, and (ii) it maintains the other advantages that OPENET4LD and its Petri nets-based model provides.

Certainly the implemented GUI, that supports the visual model, offers the option to save designed UoLs in XML format of IMS LD. So, is it justified the definition of correspondences between the visual model and the Petri nets considering that OPENET4LD allows loading UoLs in the IMS LD format? If the interface have the sole purpose of designing UoLs, in fact it would be meaningless. However, with this interface we intend to go further and visualize the workflows execution in the visual model.

Method mappings. Figure 4 shows an example of the correspondence established between a method described with the visual model and the Petri nets of OPENET4LD. As it can be seen in the figure, the visual representation of the method is mapped to two models of Petri nets: one with the control over the method execution and the other one with the control over the coordination of its plays. We can also see a certain similarity between the visual model and the right-sided Petri net. This is because YAWL language is an extension of Petri nets, and also because both nets were designed to control the coordination

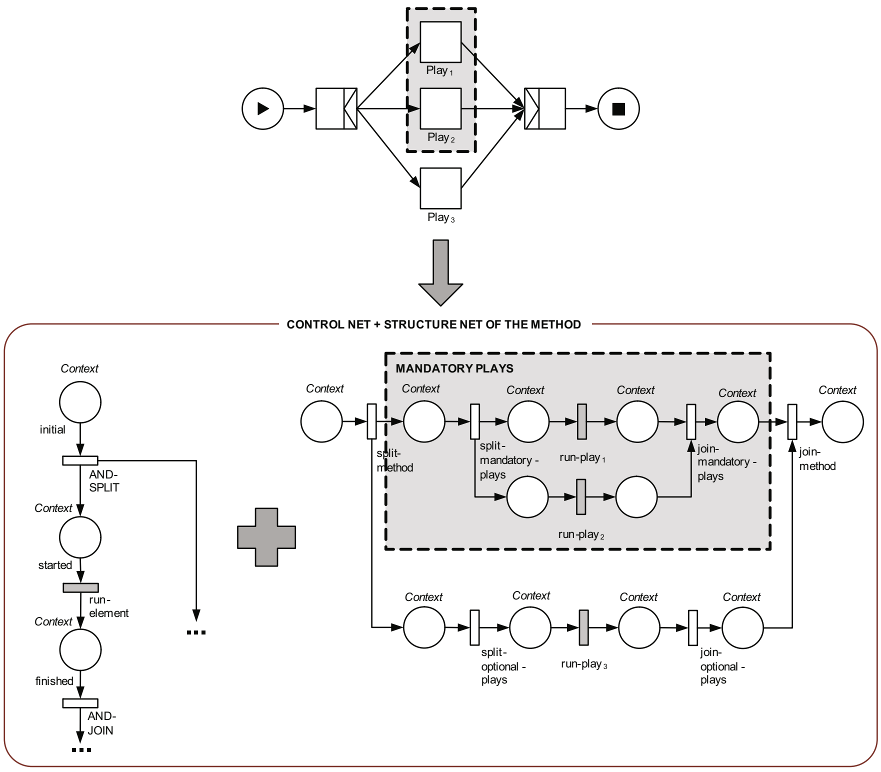


Fig. 4. Mapping between the visual representation of a method and its Petri net-based internal representation in OPENET4LD

of plays. So why do we also establish the mappings of the visual model with the right-sided Petri net? There is an important reason for this: this Petri net manages the execution of the element, and extends the IMS LD specification to support suspension, resumption or abortion of UoLs. Furthermore, it provides the means to propagate changes to the elements that depend on its execution (for example, the suspension of a play implies the suspension of all its acts, and so on). Therefore, these mappings endow our visual model with the same functionalities, so we will be able to manage the execution of the UoL from the GUI.

Play mappings. Like with methods, there is a mapping between the visual model and the two Petri nets that manage the execution of a play and coordinate its acts, respectively. In this case acts are represented in Petri net sequence.

Role-part mappings. The case of acts is different, because our visual model hides much of the structural complexity related to acts execution. In our visual model, an act is associated with a set of role-part models. However, the model does not show how these role-parts should be carried out. This association is

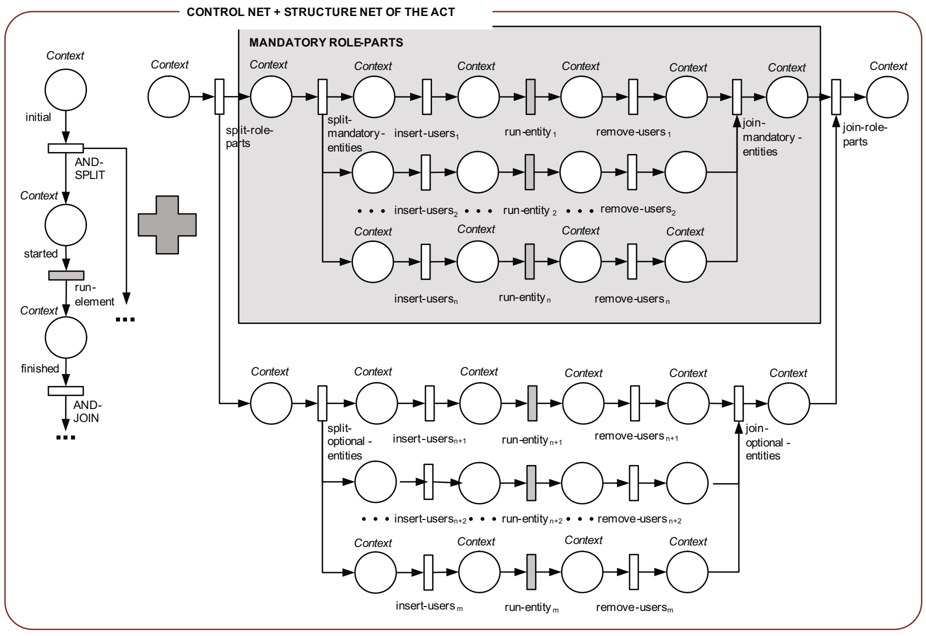


Fig. 5. Each act is mapped to these two Petri nets

done internally through the mappings, so that the Petri net that manages the execution of the act is connected to a Petri net that controls the parallel execution of each of its role-parts. For example, Fig. 5 shows the two Petri nets that control an act execution.

Each branch of Fig. 5 controls the development of the execution entity associated to the role-part. Users are inserted in the workflow execution through the *insert-users*$_i$ transitions, will run the execution entity through the *run-entity*$_i$ transition, and finally removed from the execution by means of the *remove-users*$_i$. The *run-entity*$_i$ transition represents the execution of the simple activity, structure of activities, or UoL associated wit the role-part.

The next step is the identification of the root of the last layer. For example, in Fig. 6 we show the tree of the YAWL model depicted on the top. In this tree, simple activities are the leaves nodes and the structure of activities are the branch nodes. Both node types are substituted by a control Petri net, and in the case of branch nodes, the visual element is also mapped to a net with its corresponding structure.

4 Viewer Implementation

The implementation of the GUI is depicted in Fig. 7. Its display is divided in three parts. The left part contains the panels of components and properties. The

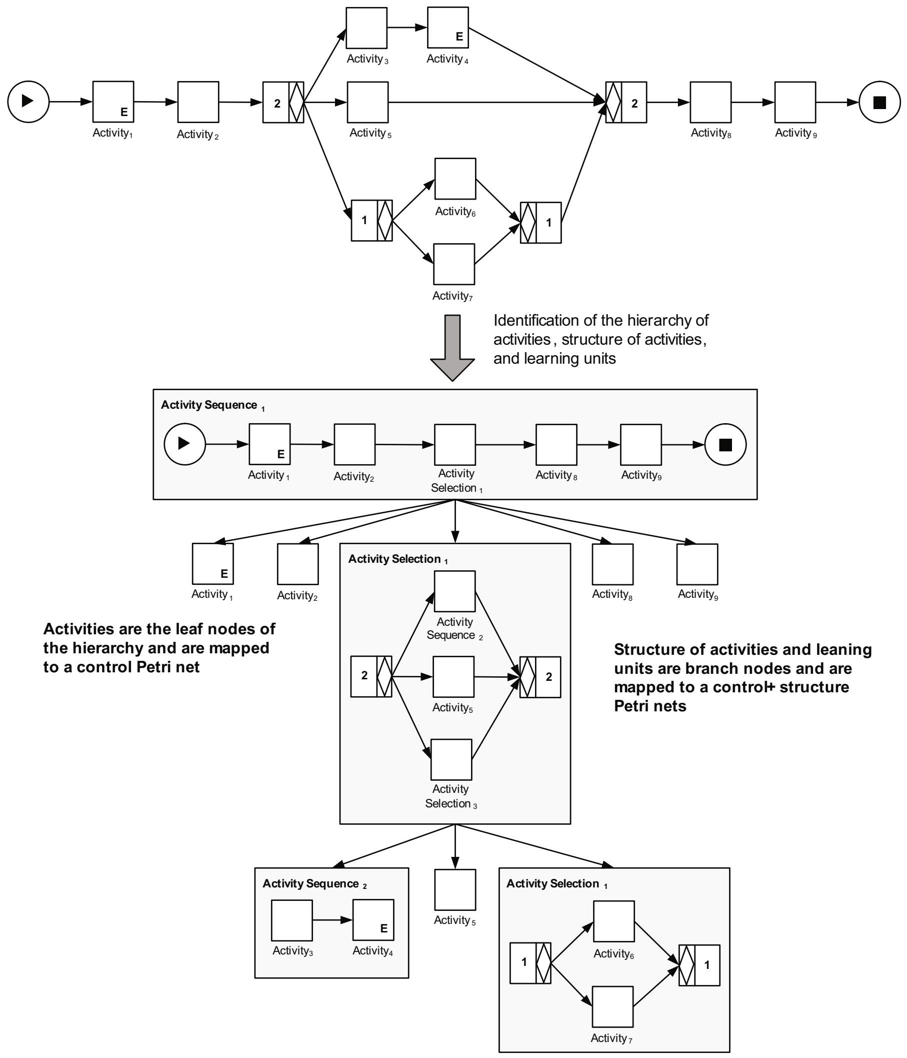
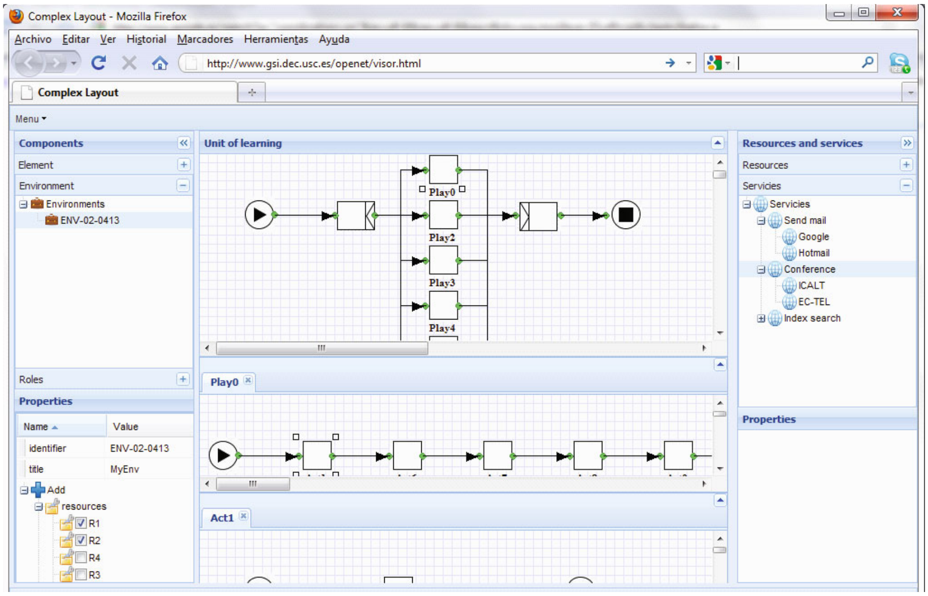

E
Activity$_1$
Activity$_2$
2
Activity$_3$
E
Activity$_4$
Activity$_5$
1
Activity$_6$
Activity$_7$
1
2
Activity$_8$
Activity$_9$
Identification of the hierarchy of activities, structure of activities, and learning units
Activity Sequence $_1$
Activity Selection $_1$
Activities are the leaf nodes of the hierarchy and are mapped to a control Petri net
Activity Sequence $_2$
Activity Selection $_3$
Structure of activities and leaning units are branch nodes and are mapped to a control+ structure Petri nets

Fig. 6. Example of hierarchical decomposition of a role-part model

first contains the visual components the designer may use to create his visual models, that is, it contains shapes representing the plays, acts, activities, control structures, roles, and environments. These components can be dragged to the central panel in order to create and annotate the visual model.

As it can be seen in the figure, the central panel is again divided in three parts which tally with the three layers of our visual model. Methods are modeled at the top, plays at the central part, while each role-parts of an act is designed in a tab at the bottom. The interaction between the three parts of the model is



Name	Value
identifier	ENV-02-0413
title	MyEnv

Add
resources
R1
R2
R4
R3

Fig. 7. Screenshot of the GUI for the design/edition of IMS LD units of learning

simple: the selection of play at the top panel will open a tab in the central part with the sequence of acts associated to the selected play; the selection of an act will open all its role-parts as tabs at the bottom part.

The properties panel is located at the bottom of left panel. Through this panel the properties of each of the selected components can be modified. For example, Fig. 7 displays the properties of the environment *MyEnv*. This panel let the designer to change the environment identifier, name, learning objects, services, and other environments it has associated.

Finally, the right side of the GUI is used to create the learning objects and services that may be associated with the environments.

5 Conclusions and Future Work

In this paper we have described a workflow-based approach that simplifies the way to create units of learning specified in IMS LD. The proposed model hides much of the complexity of IMS LD. However, this complexity is only hidden to users, since internally we maintain the structure established by IMS LD by means of a set of mappings to the Petri nets of OPENET4LD, the engine that supports the execution of our models. With this approach we take advantage of the simplified approach for designing UoLs, but maintaining the advantages that OPENET4LD and its Petri nets-based model provides.

As future work we plan to complete (i) our visual model to support the level B of IMS LD, and (ii) the GUI to support the execution of the units of learning in a workflow-oriented way.

References

1. IGL Consortium: IMS Learning Design Information Model, Version 1.0 Final Specification (2003)
2. Griffiths, D., Beauvoir, P., Sharples, P.: Advances in editors for ims ld in the tencompetence project. In: Proceedings of the 8th IEEE International Conference on Advanced Learning Technologies (ICALT 2008), Santander, Spain, pp. 1045–1047. IEEE Computer Society (2008)
3. Gutiérrez Santos, S., Pardo, A., Delgado Kloos, C.: Authoring courses with rich adaptive sequencing for ims learning design. J. UCS 14(17), 2819–2839 (2008)
4. Hernández Leo, D., Villasclaras-Fernández, E.D., Asensio-Pérez, J.I., Dimitriadis, Y.A., Jorrín-Abellán, I.M., Ruiz-Requies, I., Rubia-Avi, B.: Collage: A collaborative learning design editor based on patterns. Educational Technology & Society 9(1), 58–71 (2006)
5. Laforcade, P.: Graphical representation of abstract learning scenarios: the uml4ld experimentation. In: Proceedings of the 7th IEEE International Conference on Advanced Learning Technologies (ICALT 2007), pp. 477–479. IEEE Computer Society, Niigata, Japan (2007)
6. Neumann, S., Oberhuemer, P.: User evaluation of a graphical modeling tool for ims learning design. In: Spaniol, M., Li, Q., Klamma, R., Lau, R.W.H. (eds.) ICWL 2009. LNCS, vol. 5686, pp. 287–296. Springer, Heidelberg (2009)
7. Martínez-Ortiz, I., Moreno-Ger, P., Sierra-Rodríguez, J.L., Fernández-Manjón, B.: A flow-oriented visual language for learning designs. In: Li, F., Zhao, J., Shih, T.K., Lau, R., Li, Q., McLeod, D. (eds.) ICWL 2008. LNCS, vol. 5145, pp. 486–496. Springer, Heidelberg (2008)
8. Dodero, J.M., Martínez del Val, Á., Torres, J.: An extensible approach to visually editing adaptive learning activities and designs based on services. J. Vis. Lang. Comput. 21(6), 332–346 (2010)
9. Karampiperis, P., Sampson, D.: Towards a common graphical language for learning flows: Transforming bpel to IMS learning design level Arepresentations. In: Proceedings of the 7th IEEE International Conference on Advanced Learning Technologies (ICALT 2007), pp. 798–800. IEEE Computer Society, Los Alamitos, CA, USA (2007)
10. Vidal, J.C., Lama, M., Sánchez, E., Bugarín, A.: Application of petri nets on the execution of ims learning design documents. In: Proceedings of the 3rd European Conference on Technology Enhanced Learning (EC-TEL 2008), pp. 95–100. Springer, Maastricht, Holand (2008)
11. van der Aalst, W.M.P.: The application of petri nets to workflow management. The Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
12. Ellis, C.A., Nutt, G.J.: Office information systems and computer science. ACM Comput. Surv. 12(1), 27–60 (1980)
13. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data Knowl. Eng. 47(2), 237–267 (2003)

14. van der Aalst, W.M.P., Hofstede, A.H.M.: Yawl: Yet another workflow language. Information Systems 30(4), 245–275 (2005)
15. Vogten, H., Martens, H.: CopperCore Version 3.3. Open University of Netherlands (2009)
16. Escobedo, J.P., de la Fuente Valentin, L., Gutierrez, S., Pardo, A., Delgado Kloos, C.: Implementation of a learning design run-time environment for the .LRN learning management system. Journal of Interactive Media in Education 1, 1–12 (2007)
17. Paquette, G., Léonard, M.: The educational modeling of a collaborative game using mot+ld. In: Proceedings of the Sixth IEEE International Conference on Advanced Learning Technologies (ICALT 2006), pp. 1156–1157. IEEE Computer Society, Kerkrade, The Netherlands (2006)
18. Dodero, J.M., Ghiglione, E.: Rest-based web access to learning design services. IEEE Transactions on Learning Technologies 1(3), 190–195 (2008)