OPENET LD
An Ontology-based Petri Net Engine to Execute IMS LD Units of Learning

Juan C. Vidal, Manuel Lama, Eduardo Sánchez, Alberto Bugarín, Adrián Novegil
Department of Electronics and Computer Science
University of Santiago de Compostela
Santiago de Compostela, A Coruña, Spain
{juan.vidal, manuel.lama, eduardo.sanchez, alberto.bugarin.diz}@usc.es, adrian.novegil@rai.usc.es

Abstract—In this paper an ontology-based Petri net engine to execute units of learning based on the IMS Learning Design (IMS LD) specification is presented. This engine has a layered architecture that enables the execution of IMS LD learning flows by means of high-level Petri nets representing the semantics of the IMS LD dynamic elements. Even if changes in the specification were made, the engine would not have to be re-implemented, and only the introduction of new rules and/or axioms in the ontology would required. Moreover, a service-oriented architecture has been implemented for allowing web-based learning management systems to access to the IMS LD engine.

Keywords—petri nets; ontologies; learning flow; IMS LD

I. INTRODUCTION

In recent years an important effort to develop engines and players to execute IMS Learning Design (IMS LD) [1] units of learning has been made. In these developments the learning flow execution is directly coded in a programming language [2,3,4], which makes difficult to adapt to possible changes in the semantics of IMS LD.

To overcome this limitation, some approaches translates the IMS LD learning flow model into a workflow specification language [5,6]. However, these works (i) do not define a computational formal model describing the learning flow that represents the coordinated execution of the learning activities, which impedes to check the consistency of the learning flow (deadlock-free issues, for instance), and (ii) the language to which the IMS LD is translated cannot represent some elements of the learning flow model, such as the conditions and variables of the IMS LD level B.

In this paper we present OPENET LD, an ontology-based Petri net engine for the execution of units of learning based on the IMS LD specification level A. This engine has a layered and flexible architecture that combines three ontologies [7] to represent the knowledge needed to carry out a semantic validation, and to execute IMS LD units of learning. Two of these ontologies contain a declarative description of the knowledge required for executing hierarchical high-level Petri nets (HLPN) [8], and the third ontology represents both the semantics of the IMS LD elements and the mappings between each element and the HLPN instance describing how it must be executed. Under this approach, a formal and expressive model to execute IMS LD is provided (Petri nets formalism) and changes in the semantics can be declaratively made through the ontology (without reprogramming the engine).

II. PETRI NET-BASED MODELING OF IMS LD

In our approach, each learning flow element of the IMS LD, and particularly every method, plays, acts, and activity structures, is represented through a HLPN describing its execution semantics, including the conditions in which the flow element is finished, stopped, and resumed. The complete specification of a learning design will be a hierarchical HLPN that represents the complete semantics of the IMS LD execution model. To illustrate this approach we will explain how the method element of the IMS LD is executed, but a more detailed discussion of the HLPN modeling of the IMS LD elements can be found in [9].

A method describes the dynamics of the learning process and it is composed of a number of plays. These plays can be interpreted as the run-scripts for the execution of the unit of learning and have to be executed concurrently and independently of each other. Following this description, Fig. 1 depicts the HLPN that models the execution of a method, showing how each of the method attributes (with a dynamic behavior) is related either to a place or a transition of the Petri net. The AND SPLIT transition enables the concurrent execution of a set of parallel branches where each of these branches models the execution of a play. Thus two set of branches can be distinguished:

- The boxed branches represent the plays that must be executed mandatory; that is, it states that a unit of learning is completed when this set of play(s) is (are) completed.
- The other branches represent optional plays.

The AND JOIN transition synchronizes the execution of the mandatory plays, so that when they all have finished its execution (i) a token is produced in the when-play-completed place; and (ii) another one on each of the stopping places. These last tokens will enable the stopping of the optional plays. Once all of them have finished (or stopped) their execution, the AND JOIN can fire and thus a token could be moved to the final place indicating the end of the method execution.
III. ARCHITECTURE OF THE IMS LD ENGINE

The IMS LD engine\(^1\) has an open and modular software architecture composed of three ontology-based layers (depicted in Fig. 2) that capture the semantics for the execution of units of learning modeled with Petri nets. Using ontologies to represent declaratively the knowledge to execute IMS LD facilitates the modification of the semantics without recoding the engine.

A. HLPN Engine

The IMS LD engine has been built on the top of a HLPN executor based on an ontology that represents semantically the static and dynamic models of a Petri net. This ontology thus describes the behavior of HLPN and the reasoning methods from which the engine controls the execution. This engine consists of the following components (Fig. 2):

- **Schema.** This module contains the elements of the HLPN ontology [11]; that is, the concepts, attributes, and axioms that represent formally a Petri net.

- **Data.** The module stores the instances of the HLPN ontology components, that is, it contains the Petri nets that represent the different elements of the IMS LD learning flow.

- **Rules.** This facade contains rules that enable (i) to create, modify, and delete instances of the Petri nets ontology, and (ii) to automate the execution of the net, obtaining the evaluations of terms, querying for the concurrent transition modes, firing a transition mode and setting the new state of the Petri net, etc.

- **FLORA-2 Reasoner.** FLORA-2 [12] is a knowledge base engine that extends classical predicate calculus with the concepts of objects, classes, and types, and it also integrates logic programming and deductive databases with the object-oriented programming paradigm. It uses a F-Logic dialect as the language to create knowledge bases and ontologies.

- **Reasoner Interface.** The methods of this interface translate the Java objects used to manage the Petri nets into its F-Logic representation in the ontology.

- **HLPN Engine Interface.** It is a Java facade responsible for the management and execution of HLPNs. It implements several ways of inserting HLPN in the knowledge base (PNML and F-Logic formats), and handles the execution of HLPN by means of an execution manager module that schedules the active transition modes to fire in an execution step. This execution manager supports the execution of Java methods, web services, and user tasks.

---

\(^1\) http://www.gsi.dec.usc.es/hlpno/downloads.html
B. HHLPN Engine

The second vertical layer of the IMS LD engine in Fig. 2 defines the semantics of Hierarchical HLPN (HHLPN). With this kind of nets it is possible to build a large net (the HLPN that models the complete IMS LD learning flow) by combining a set of smaller nets (the HLPN that models each IMS LD component). Therefore, this layer allows the creation of the Petri net that will execute the learning flow of the IMS LD. The main components are:

- **Schema.** This module stores the components of the ontology that describes the semantics of HHLPNs. This ontology only uses two mechanisms for Petri net composition: node substitution and node fusion.
- **Data.** This module stores the instances of the pages (flat Petri nets) that compose the hierarchical net and the fusions and substitutions that define how these pages are related between them.
- **Rules.** This set of rules enables to (i) create, update and delete HHLPNs; (ii) create net pages; and (iii) establish the mappings that relate the hierarchical net with a HLPN, allow to automate the creation of the flat net.
- **HHLPN Engine Interface.** It is a Java facade that provides methods (i) to obtain the complete (flat) Petri net that models the execution of the IMS LD learning flow; and (ii) to manage the hierarchical nets associated to the IMS LD components.

C. IMS LD Engine

This layer (third vertical column in Fig. 2) is based on an ontology [13] that describes the semantics of every element of the IMS LD specification. In this layer methods and rules are provided to create the instances of the unit of learning in the F-Logic language, to generate the Petri nets associated to the learning flow components, and to control their execution. The elements of this layer are:

- **Schema.** This module stores the set of classes, properties, and axioms that describe the IMS LD specification. More specifically, the axioms enable the semantic and automatic validation of units of learning.
- **Data.** This module stores the instances of ontology of IMS LD, that is, it contains the units of learning imported by the engine.
- **Rules.** This set of rules allows to (i) create, update and delete all the IMS LD components of the units of learning, (ii) create the Petri nets that capture the semantics of each IMS LD learning flow element, and (iii) compose the hierarchical net from these Petri nets. Each IMS LD learning flow element is associated with a hierarchical Petri net, and since hierarchical nets have a mapping to a flat Petri net, learning flow elements are transitively related to a flat net.
- **IMS LD Interface.** This Java facade integrates the Java interfaces of hierarchical and high-level Petri nets, allowing clients to load and execute IMS LD units of learning. Taking into account that the execution semantics of IMS LD is defined for flat Petri nets, this interface uses the HHLPN engine interface to combine the Petri nets that define the structure of the unit of learning, and then flatten the hierarchical net to a HLPN. The execution of this net
means the execution of the IMS LD and will be controlled by the execution manager defined in the HLPN engine interface.

To evaluate the consistency and correctness of the IMS LD engine we have developed a graphical interface (Fig. 3) that shows the execution state of the learning flow components, and enables users to modify its state. This evaluation has been made with 21 units of learning based on the IMS LD level A specification.

IV. SERVICE-ORIENTED ARCHITECTURE

In order to facilitate the use of the IMS LD engine by learning management systems, which are typically available through web interfaces, we have developed a service-oriented architecture that externalizes the main capabilities of the engine through a set of web services. Particularly, the services available are (Fig. 4):

- **UoL2Flora.** Set of services that translate the IMS LD units of learning from XML-Schema format into the F-Logic implementation of the ontology used by the engine. These services use the PNDefinition and IMS2PN web services to create the Petri nets which are associated to the IMS LD components.

- **UoLValidation.** Set of web services that carry out a semantic validation of the semantics IMS LD units of learning published by the engine. The semantics is captured in the ontology axioms, and validation services query for instances that are not compliant with these axioms.

- **UoLExecution.** Web services that enable clients to recover a specific unit of learning instance from the Petri net repository, and load it into the engine. This service uses the PNExecution service to create an execution instance of the complete HLPN.

- **UoLStateManagement.** Set of services that enable clients to manage the components state of a unit of learning through the HLPN state associated to such component. They allow to (i) query about the state of the IMS LD learning flow, and (ii) carry out actions (stop, resume, select, and execute) that modify the state of the IMS LD components. These services internally invoke the PNStateManagement web services to control the state of the Petri nets underlying the execution of IMS LD units of learning.

- **UoLPublication.** Set of web services that allow to publish IMS LD units of learning. This publication means the translation and semantic validation of the unit of learning as well as the storage in the repository managed by the engine. These services will use the UoL2Flora and UoLValidation services for their operations.
V. CONCLUSIONS

The combination of ontologies and high-level Petri nets as the basis for developing the IMS LD engine has clear advantages: first, with Petri nets we have a formal and expressive model to represent the semantics for the execution of IMS LD; second, with the use of ontologies to represent both Petri nets and IMS LD specification, the knowledge is declaratively integrated into the engine. Thus new rules and axioms can be directly integrated without modifying the Java facets of the engine (no reprogramming is needed). This feature gives the IMS LD engine with the capability of adapting its operation to the semantics of the specification.

Currently, we are extending the IMS LD engine to be able to manage units of learning based on the level B of the specification. For it, (i) IMS LD Java facade must be modified to include the new concepts of the level B; and (ii) IMS LD engine interface must be changed to generate HLPNs able to evaluate level B conditions. However, the semantics for executing the units of learning (underlying in the HLPNs) will not need to be modified.

ACKNOWLEDGMENTS

Authors wish to thank the Ministerio de Industria, Turismo y Comercio and the Ministerio de Educación y Ciencia for their financial support under the projects TSI-020301-2008-19 and TSI2007-65677C02-02 respectively.

REFERENCES