Adaptation in a PoEML-based E-learning Platform

Roberto Perez-Rodriguez, Manuel Caeiro-Rodriguez, and Luis Anido-Rifon
Department of Telematics
University of Vigo
Vigo, Spain
rperez@gist.det.uvigo.es, mcaeiro@det.uvigo.es, lanido@det.uvigo.es

Abstract—**In EML-based e-learning platforms for engineering education, adaptation can be focused whether in the EML meta-model or in the run-time environment. In the approach centered in the EML meta-model, adaptation is carried out by advanced modeling and late modeling. In the approach based on the run-time system, it is the run-time system the one that supports the change types that are allowed. This paper presents a conceptual framework and software architecture aimed to support late modeling in an e-learning system based on PoEML (Perspective-oriented Educational Modeling Language).**

***Keywords*— CSCL, adaptation, late modeling**

I. INTRODUCTION

In the last years, it has been developed a new approach to e-learning systems design, based on Educational Modeling Languages (EMLs). Those languages enable the definition of a Unit of Learning (UoL) independently from the final system used for delivering the UoL, enabling thus a certain degree of reuse in the design of UoLs [1]. This approach supports the definition of UoLs in a pedagogy-independent way.

Unlike both implicit and explicit models that manage contents, an EML not only describes the static content structure of a UoL, but it also accurately describes dynamic issues such as activities to perform, composition of groups of participants, the order between activities, etc. Therefore, the description of a UoL implies the description of a process. The process described by an EML can be of collaborative nature, that is, several participants collaborate in order to achieve a common goal. That is a difference from languages as SCORM (ADL) [2], which only allows for the definition of processes in which only one participant is involved.

The PoEML language [3] is within this conceptual framework. This modeling language has been designed following a separation-of-concerns approach, encapsulating thus quasi-independent parts of a UoL model in separate aspects/perspectives.

The UoLs described by an EML have a life cycle that can be decomposed into the following phases:

- Authoring: in this phase, the UoL is designed by making use of an authoring tool. This phase is also known as design-time.
- Publication: in this phase, the UoL is imported into an execution engine.
- Delivering: in this phase, the participants perform the UoL. This phase is also known as run-time.

In the Figure 1, it is shown how the author of a UoL publishes it into a Learning Management System (LMS) after the authoring phase.

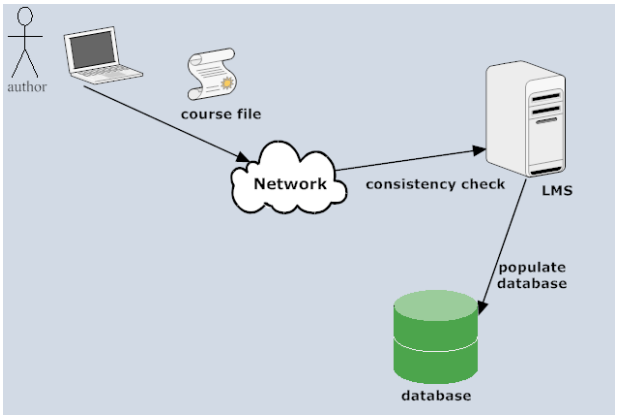

Figure 1. Authoring and publishing of a UoL.

In systems based on EMLs, methods for promoting learning are in the form of structured processes. Collaboration systems based on processes, whose most representative developments are the WfMSs [4], indicate a process specification in which the activities to be performed by the different actors are defined. Due to their characteristics, those systems are very similar to EMLs, and therefore they are relevant for this study.

Some works have proposed to transform processes modeled with an EML to processes modeled with a workflow language [5] [6]. In this work [7], an EML is even used as a true workflow language. Because of the reasons already mentioned, we will refer to EML-based e-learning systems as a type of process-based system.

A main issue in EML-based e-learning systems is that of the adaptation [8]. Late modeling allows designers to leave some parts of the UoL unmodeled until run-time. Often, designers may prefer to model the most relevant parts of the UoL, remaining the rest to be modeled at run-time. Therefore, it is required to support incomplete (or abstract) models, which will be specified during run-time.

This paper introduces both a conceptual framework and software architecture for supporting late modeling of PoEML learnflows.

II. COMMON APPROACHES TO ADAPTATION

From a technical point of view, there are some mechanisms that are used to carry out adaptation. In order to support adaptation in process-based systems, it is necessary to take into account some aspects both from the definition of processes (design-time) as well as from the execution of processes (run-time). Next, we discuss those approaches following the classification in [9]:

- Approach centered in process definition meta-model
  - Advanced modeling
  - Late modeling
- Approach centered in the execution system
  - Type adaptation (evolutionary change)
  - Instance adaptation (ad-hoc change)

A. Approach centered in process definition meta-model

In the approach centered in the process definition meta-model, it is the meta-model the one that determines the structure and the allowed types of changes. The following lines are considered: advanced modeling and late modeling.

Advanced modeling allows authors to consider alternatives during design-time. In this case, the different alternatives (learning paths) that are allowed during execution are modeled in the process definition at design-time. Despite the fact that a UoL has well-defined objectives, those can be achieved by different routes, depending for example on the student's performance. The support of this requisite is more complex if we consider alternatives in every concern that can be modeled (data, participants, time-constraints, etc). Therefore, an EML might allow for the modeling of alternatives in the different concerns involved in a UoL. That modeling of alternatives should be done in a manageable and compact way.

During run-time it should be possible to select one of those alternatives. There are two main ways to do the selection [10]:

- In accordance with the state of execution. Using conditions that are evaluated in specific points (e.g. at the start of an activity) or events that that happen at any time. The state can be related with the characteristics of participants, results obtained in past activities, etc.
- In accordance with the decision of an authorized participant. In traditional education, often teachers make decisions on the best way to achieve learning objectives.

Late modeling allows for leaving some parts of the design without being modeled, in order to be modeled during run-time. Some parts of the process definition are leaved as black-boxes, in order to be modeled during execution. In [10] late modeling is dealt with as a dynamic refining of the models. Often, some parts of a UoL may not have a clear solution and it may be not possible to model them. Designers may prefer to model the most relevant parts and leave without modeling the less relevant ones until run-time. Therefore, it is required to support incomplete (or abstract) models that will be specified during run-time.

B. Approach centered in the execution system

In the approach centered in the execution system, it is the execution system the one that supports the types of change that are allowed. The following lines are considered: instance adaptation and type adaptation.

Instance adaptation is usually named exception management. An exception can be seen as an occasional deviation from the normal behavior of a process. Exception management is needed for dealing with the deviation in run-time from the execution plan foreseen during design-time [11]. In the e-learning field, an exception can be produced when the model is deficient in some realization of the process, as when a certain route (learning path) is over dimensioned respect to other learning paths, causing in this way a competitive disadvantage to the learner that has followed the over dimensioned route. In that case, exception management is in charge of correcting that wrong situation, without need for stopping all process instances, fix the model, and enact all the instances again.

Type adaptation deals with process instance migration during a process execution from an old schema to a new one. One of the main problems in this approach is to make a migration of the process execution state to the new one [12]. Type adaptation can be presented as a dynamic evolution of the model, that is to say, the UoL model described with an EML is able to evolute in time to adapt itself to new situations, for example: the change in a study planning, more learners that expected, a new approach to teach some contents, etc.

III. A LATE MODELING EXAMPLE OF A POEML LEARNFLOW

Educational Modeling Languages such as PoEML have a static view of the world, and they do not support evolutionary and dynamic changes. In the literature, there are many works addressing the lack of flexibility and/or of adaptation in EML-based e-learning systems, such as [13] [14].

Figure 2 shows an example of collaborative practice formalized with PoEML. In this example, it can be seen how the work is divided into three independent activities (corresponding to each one of the Newton's three laws of motion), which are assigned to three different participants. Each activity is reviewed by a different participant. When the reviews of the three participants are positive, the three participants have to edit collaboratively a document that reflects the work performed in the previous three activities. Finally, the teacher makes an evaluation of the last activity, resulting in one of three possible flows (finish, minor revision, major revision). The example learnflow can be carried out just by using a wiki engine for edition and basic HTML forms for assessment by the teacher.

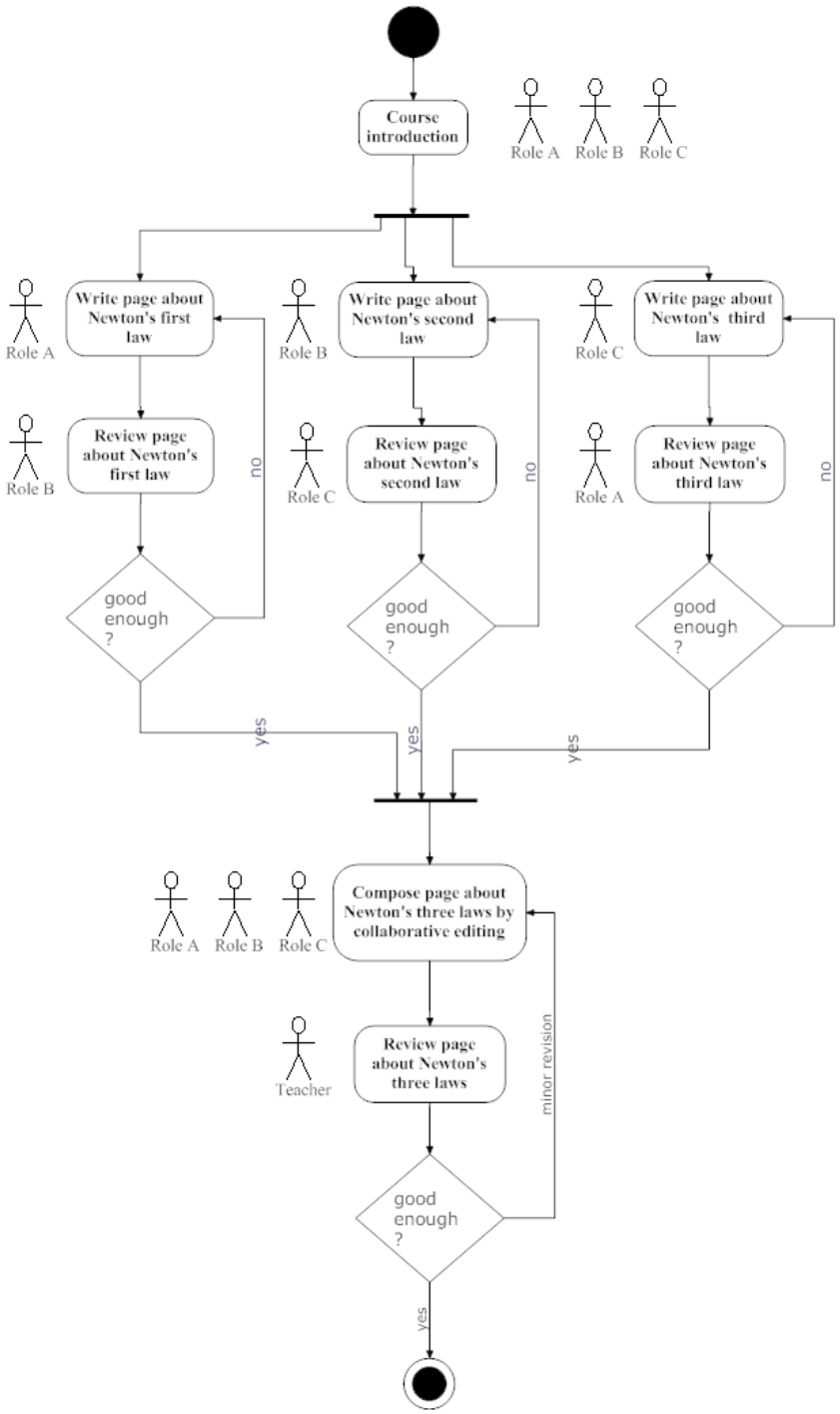

Figure 2 A PoEML learnflow

A collaborative activity is shown in Figure 3. It is worth noting that the edition of a page in a wiki is a kind of activity with no explicit end. In this case the end of the activity will be the temporal deadline. Using PoEML as the modeling language, this collaborative activity is modeled as a `educational scenario` in which there are three roles involved.

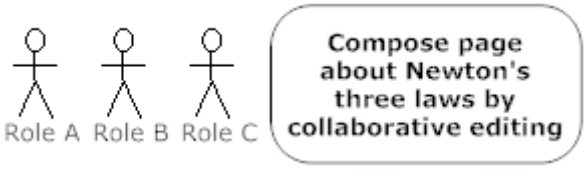

Figure 3. Collaboration at an activity level

The graphical flow-based representation in the Figure 2 is useful to put into manifest the sequencing of operations in a learnflow. Notwithstanding, we will follow an aspect-oriented approach in which the different aspects that compose a learnflow are treated in a quasi-independent way. Thus, in the learnflow of the Figure 2 several aspects can be identified:

The initial `scenario` is shown in Figure 4.

Figure 4. Initial scenario in the learnflow

Once the initial `scenario` is finished, three `scenarios` are launched in parallel. In the Figure 5 it is shown one of those three parallel `scenarios`.

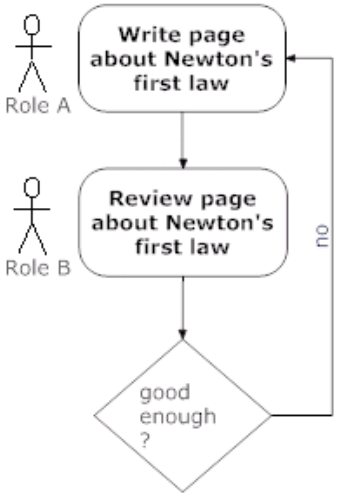

Figure 5. One of the three parallel scenarios in the learnflow

Finally, participants are requested to perform a collaborative activity, which is evaluated by the teacher. Figure 6 shows that final `scenario`.

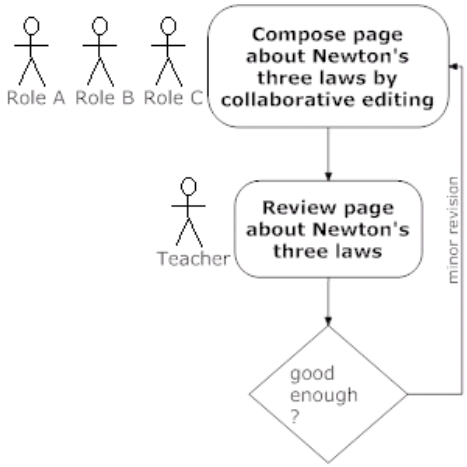

Figure 6. Final scenario in the learnflow

The ordering of `scenarios` is shown in Figure 7.

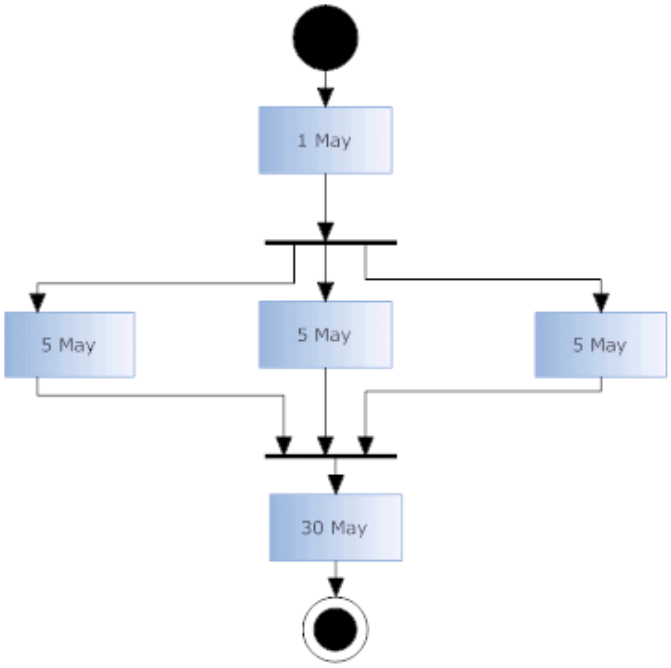

Figure 7. Order and temporal view of scenarios

In the activity *Write page about Newton's first law*, the work for the assigned participant can be specified in more detail. For example, the content of the fist law page might contain: formulation, history, and example of application. In order to leave unspecified the nuts and bolts of the activity *Write page about Newton's first law* in run-time, a *placeholder* activity must be placed during design-time. In this way, the content of the placeholder activity can be specified during run-time, and thus adapted to both teacher's and learners' preferences. With this approach it is obtained a good grade of freedom in design.

In design-time it is just not possible to foresee the behavior of learners. It is necessary to wait until run-time to monitor the learners' reactions and performance. The teacher may want to tune the model in some ways:

- Adding more activities
- Changing the temporal constraints for activity finishing
- Adding/skipping a `goal`, etc.

In the next section we will see the procedure for adding/editing/removing elements of the model.

IV. LATE MODELING OF POEML LEARNFLOWS: AN ASPECT-ORIENTED APPROACH

Following an aspect-oriented approach, the allowed changes in the model can be systematically categorized in *aspects*, each one dealing with a specific concern. In Table 1, it is shown a systematic categorization of run-time changes.

In the literature there are some works such as [13] aimed at providing run-time adaptation in an aspect-oriented way. In the referred works, the underlying model is not aspect-

oriented, unlike the PoEML case. So, the PoEML aspect-orientation is a grant for easier aspect-oriented changes during run-time.

A. *Situating placeholders in the UoL model*

Many times, the complete structure of a UoL cannot be specified during design-time. In that case, it is desirable to count with late modeling support, which provides the needed flexibility to model the remaining parts of a UoL at run-time.

The approach here is to use a special type of construct, named *placeholder*. The placeholder occupies the place of the unmodeled `element` or `specification`. Thus, there will be as many placeholders as types of constructs in the EML at use. In the PoEML case, we identify the following placeholder types (Table **1**), in accordance with the `educational scenario` meta-model in Figure 8. This table puts into manifest the aspect composition feature in our approach. The `scenario` placeholder may be composed of placeholders for `goals`, `environments`, `order specification`, and `temporal specification`. In the same way, the `environment` placeholder contains at least a `tool` placeholder.

Placeholder type	Contained placeholders
Scenario placeholder	Goal placeholder, environment placeholder, order placeholder, temporal placeholder
Goal placeholder	
Environment placeholder	Tool placeholder
Tool placeholder	
Order placeholder	
Temporal placeholder	

Table 1. Placeholder classification

The `scenario` placeholder is a black box that reserves space for a `scenario` model. The language constructs will be added at run-time in the space kept by the placeholder.

The `goal` placeholder reserves the space for the definition of a learning objective.

The `data` placeholder is used for reserving the space to add resources and/or activities to a UoL at run-time.

The `order` placeholder keeps the space for adding ordering constraints at run-time.

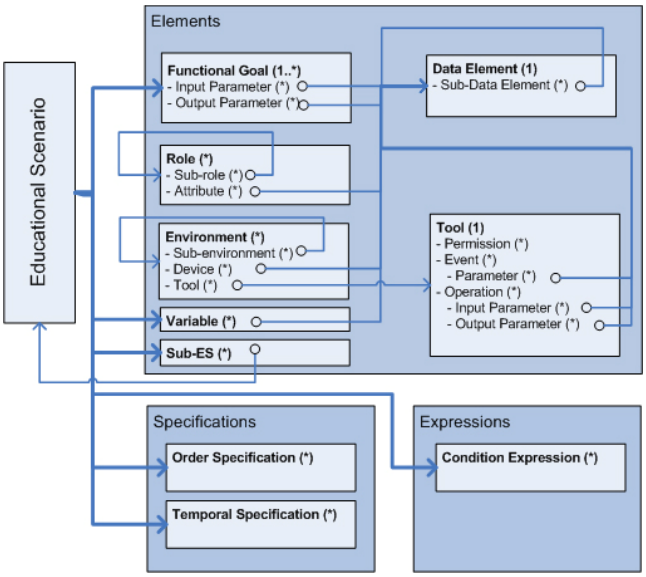

Figure 8. Elements in an Educational Scenario

The `temporal` placeholder enables the deferred definition of temporal constraints in the Unit of Learning.

Finally, the `participants'` placeholder can be situated in the place of the complete specification of `participants'` roles in the UoL.

B. *Modeling black-boxes (placeholders) at run-time*

When in a learnflow instance the execution flow reaches a black box, the `educational scenario` is instantiated, but it cannot progress to the `accessible` state. Figure 9 shows a view of a learnflow with a `temporal` placeholder in the final `scenario`.

Figure 10 shows the allowed execution states for a `scenario` instance. When a `scenario` contains a placeholder, the scenario instance remains in the `not accessible` state.

At run-time, the teacher can fill the placeholders with activities, temporal constraints, etc. by making use of the authoring environment. When the teacher commits a change (the filling of a placeholder) the related `scenario` can progress to the `accessible` state.

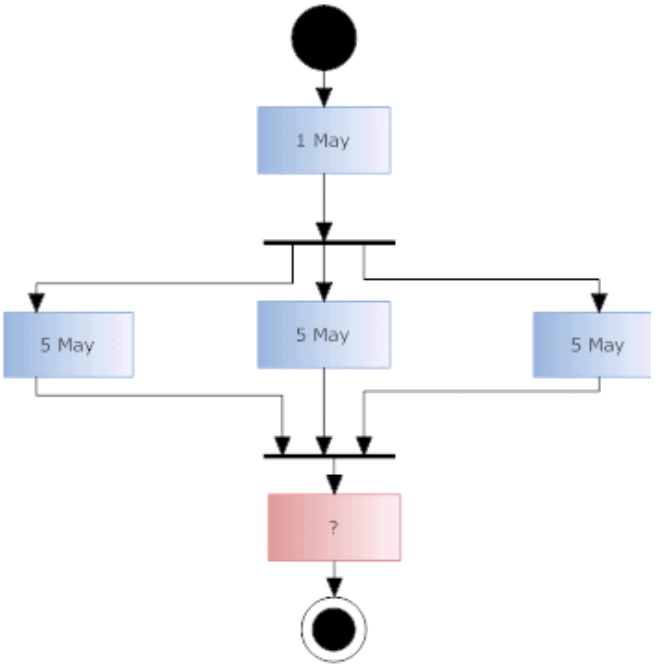

Figure 9. Temporal placeholder in the final scenario

C. Updating of instances

The model is the generic/abstract description of a PoEML learnflow; it contains the formal manifest of elements (`scenarios`, `goals`, `participants`, etc.) that make part of a learnflow. Every learner who starts a learnflow enacts a new instance. So, there are as many instances as participants in the learnflow.

Every learnflow instance has its own state, which depends on the state of the `goals` that the instance contains. In the same way, the state of a `goal` depends on the state of its input and output constraints, as well as on its dependent sub-`goals`.

When a teacher commits a change in the learnflow model, every learnflow instance has to be re-evaluated from root to leaves in order to update its state.

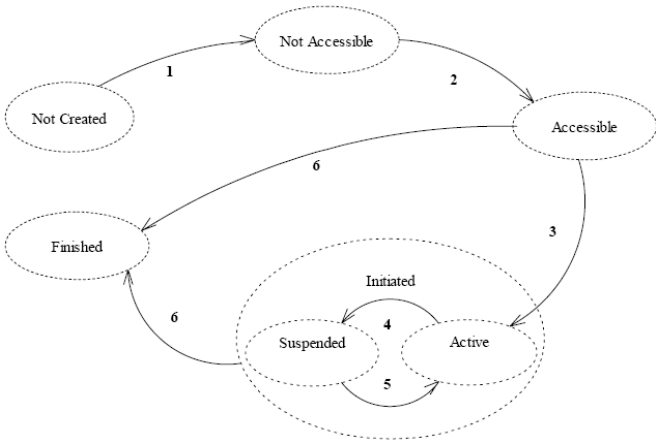

Figure 10. Execution states of a `scenario` instance

V. THE MOODLE EXTENSION

We have developed a Moodle [15] extension that is composed of a new course type with three views: run-time, authoring tool, and monitoring. The focus of this paper is the authoring tool, but for its correct exposition we need to detail the run-time and monitoring views.

In current Moodle course types, there is needed just one database schema to store the course description. In our case, we need a database schema for course descriptions and another one for run-time course instances.

A. The authoring environment

The authoring process consists on a series of atomic operations. In the set of possible operations there are ones such as:

- Add/edit/delete a Scenario
- Add/edit/delete a Goal
- Add/edit/delete a resource/activity

Atomic changes are validated in order to assure they are consistent with the overall course design. After verifying its consistency, the atomic change is committed against the database schema that contains course descriptions. The granularity level can be set as desired: an atomic operation may be to add a new Goal with all the required fields, or may be to edit a previously created Goal and change just one field. The authoring process consisting on atomic changes presents some valuable advantages:

- Atomic changes are automatically seen by other co-authors
- There is no need for a complex consistency check like the one needed when importing a previously created manifest file that contains a full course description

The consistency check commented in the last paragraph is yet necessary for interoperability purposes. Course description follows PoEML data model. Therefore, course descriptions can be exported from their database schema into XML manifest files. Hence, a XML manifest file that contains a course description can be imported into another PoEML-compliant Learning Management System.

Several experts are able to work collaboratively on the same design. The co-authors make atomic changes in the course design, and those changes are committed against the database schema for course descriptions. The course design can be tested on-the-fly by creating a new course instance and testing it in the run-time environment. The Figure also shows that manifest files containing course descriptions can be imported and exported into the LMS Engine database.

B. The run-time environment

Figure 11 shows a screenshot of the run-time environment for a participant in a course. It consists on tree course views: Structural view, Scenarios tree view, Goals view, and Functional view. We start the exposition with the structural view.

When a participant is enrolled in a course, a course instance is assigned to that participant. At that time, the course structural view is displayed at the browser. The main parts of that page are the navigation bar (Ariadna's thread), the `goals`, the `environments` and the sub-`scenarios`.

The navigation bar or Ariadna´s thread shows at any time the aggregation level in which the participant is. The `goals` box shows all the `goals` in the current `scenario`. The color of each `goal` is an indication of its state. The possible states are: Not Proposed, Not Attemptable, Attemptable, Pending, Completed, and Failed. Each `goal` box can be expanded in order to see information about `input/output constraints`, as well as `input/output parameters`.

The learning resources and activities are contained into one `environment` box. An `environment` contains Moodle-native resources and activities, such as quizzes, forums, and wikis. At the bottom of the page, the sub-`scenarios` are displayed.

In the Functional view, the participant can see the `goals` state and a list of all the `environments` needed to accomplish each `goal`. Depending on the `goal` that the participant is working in, the `environments` are displayed.

The Tree view and Goals View show a map of the current state of the `scenarios` and `goals` for a certain course instance. Those two views may be helpful to see which `goals` have been accomplished and which ones remain pending.

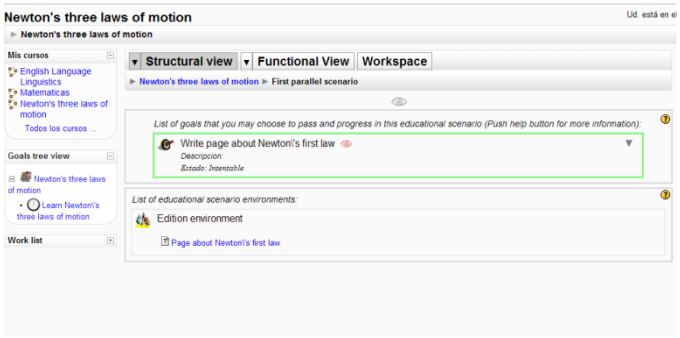

Figure 11 Screenshot of the run-time environment

C. *The monitoring environment*

The course monitoring feature is available for the teacher's role. This feature is divided into two different views:

- Monitoring per course element (element-driven)
- Monitoring of all elements per student (student-driven)

Figure 12 shows the monitoring of a `goal`. At the top of the page, we can see the possible states of the `goal` and the number of students per state. A histogram is used for showing that information in a graphical way. At the bottom of the page, we can see a table detailing the status of the goal for every student enrolled in the course.

`Variables` and `data expressions` can also be monitored. `Variables` may serve to express students' grades in a certain quiz. `Data expressions` serve to compose decisions that depend on the student's performance, for example, a decision that depends on the grade of a student in a quiz.

The other available view is the student-driven one. That view is suitable for monitoring the student's progression through the course itineraries.

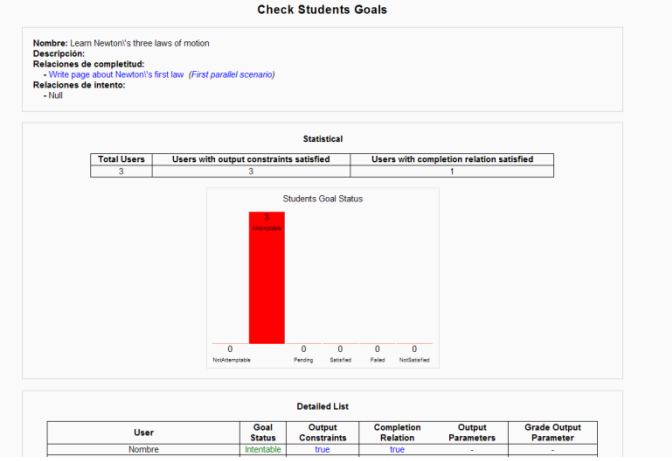

Figure 12 Screenshot of the monitoring environment

VI. IMPLEMENTATION

Figure 13 shows a diagram of the system's architecture. We detail the architecture in the following subsections:

A. *The learnflow engine (authoring, events, and information-retrieval interfaces)*

The learnflow execution engine is the core component in the e-learning system [16] [17]. The engine stores information related to `educational scenarios`, `participants`, `goals`, and rest of elements; and it makes the state of the system to evolute depending on produced events, both external and internal.

The execution engine is integrated into the e-learning system through a well-defined interface, which is based on Web Services, thus guaranteeing the connectivity requirements. At the same time, the presentation components must be as uncoupled as possible from the execution engine. That requirement entails to use a simple set of APIs. Scalability is an important requirement as well, because the execution engine is the central component of the system.

1) *Learnflow Models Manager*

The Learnflow Models Manager is the component that manages the learnflow models. This component maintains the versions of the models, and it also updates the models when required by an authorized user.

Communication from the exterior of the Learnflow Engine is made by making use of the Authoring Interface, which provides the needed service methods for authoring learnflow models. The Authoring Interface is used in run-time for the

sake of late modeling parts of learnflow models that have been specified as placeholders during design-time.

2) Learnflow Instances Manager

The Learnflow Instances Manager is in charge of managing the learnflow instances.

Communication from the presentation module is made by making use both of the Information Retrieval Interface as well as of the Events Interface.

Presentation components are able to access the execution engine in a passive way using the Information Retrieval Interface, just to get information on educational scenarios as well as on `goals`. That information retrieval is associated to navigation through `educational scenarios`:

- Get the information related to a `educational scenario`
- Get the information related to the sub-`scenarios` of a certain `educational scenario`

In a similar way, the execution engine has to manage the events from the Events Interface, which are related to the `goals` of `educational scenarios`. A participant may generate try `goal` event, with the possible outcomes of success or failure. That event generates in the execution engine the events of instantiation of `goals` that have a completion relation with the tried `goal`.

Events generated by a participant:

- Start an `educational scenario`
- Finish an `educational scenario`
- Try a `goal`, etc.

Events generated by the execution engine:

- Instantiate an `educational scenario`
- Instantiate a `goal`
- Change the state of an `educational scenario`
- Change the state of a `goal`, etc.

An event that is external to the execution engine, such as the access to an `educational scenario`, may trigger several events that are internal to the execution engine, such as the instantiation of its sub-`scenarios`. At the same time, it is imposed one restriction: the sub-`scenarios` have to contain a `goal` in the `proposed` state. That run-time behavior can be supported with Event-Condition-Action rules [18]:

- Event: a participant accesses a `educational scenario`
- Condition: the sub-`scenario` must contain at least one `goal` in the `proposed` state
- Action: to instantiate the sub-`scenario`

In summary, the interaction between presentation components and the Learnflow Instances Manager may be passive information retrieval as well as the communication of events generated by participants.

B. The middleware layer

In order to make Web Services accessible from presentation modules we make use of the functionalities provided by a SOAP [19] engine. Thus, presentation modules have an uncoupled interaction with the learnflow execution engine.

The functionality that the execution engine provides is published to a WSDL file. The service methods are for the passive information retrieval as well as for the communication of events generated by participants.

C. Presentation modules

The presentation component is composed of three subcomponents, in accordance with the provided functionality. Hence, there are three views: authoring, monitoring, and delivering.

The authoring process consists on a series of atomic operations (in case of a 'big' commit, it can always be decomposed into atomic operations). In the set of allowed operations there are ones such as: add/edit/delete a `scenario`, add/edit/delete a `goal`, and add/edit/delete a `tool`.

Atomic changes are validated in order to assure that they are consistent with the overall learnflow model. After verifying its consistency, the atomic change is committed against the database schema that contains learnflow models. The authoring process consisting on atomic changes presents some valuable advantages:

- Atomic changes are automatically seen by other co-authors
- There is no need for a complex consistency check like the one needed when importing a previously created PoEML manifest file that contains a full course description

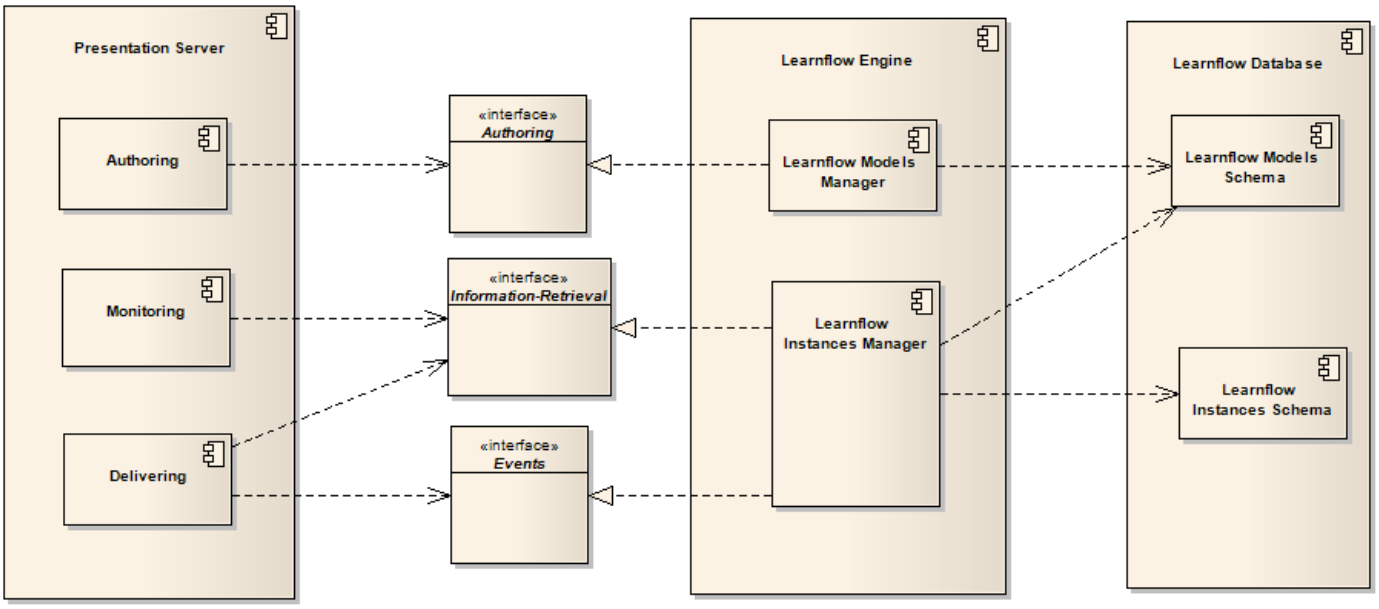

Figure 13. UML diagram of the system architecture

VII. CONCLUSIONS AND FUTURE WORK

With the PoEML-based conceptual framework described in the past sections, and the software architecture that supports it, it is enabled a kind of adaptation known as late modeling. Late modeling allows for leaving some parts of a UoL model unspecified in design-time, in order to be modeled once the learnflow is enacted, that is, during run-time.

The aspect-orientation of the PoEML language is crucial for supporting the positioning of aspect-oriented placeholders in the UoL model, thus enabling an aspect-oriented approach to late modeling.

PoEML supports well a type of adaptation based on the EML meta-model known as advanced modeling. Advanced modeling allows for considering different alternatives in the learnflow execution, known as learning paths, which are modeled during design-time.

The combination of advanced modeling and late modeling enables a high degree of adaptation, since the possibility of defining several learning paths is enhanced with the possibility of modeling parts of those learning paths during run-time.

Our future researching line is focused in the support of the adaptation types that are centered in the run-time environment: instance adaptation and type adaptation. We are currently working on providing PoEML with an execution semantics that allows for changing instances during run-time.

ACKNOWLEDGMENT

This work has been partially funded by eContentPlus program ECP 2007 EDU 417008 (www.aspect-project.org) and by the Spanish Ministry of Education and Science under grant TIN2007-68125-CO-02.

REFERENCES

[1] Rob Koper and Jocelyn Manderveld, "Educational Modelling Language: modelling reusable, interoperable, rich and personalised units of learning," *British Journal of Educational Technology*, vol. 35, no. 5, pp. 537-552, November 2003.

[2] E. R. Jones, "Implications of SCORM™ and emerging e-learning standards on engineering education," in *Proceedings of the 2002 ASEE Gulf-Southwest Annual Conference*, University of Louisiana at Lafayette, 2002, pp. 20-22.

[3] Manuel Caeiro-Rodríguez, María José Marcelino, Martín Llamas-Nistal, Luis Anido-Rifón, and António José Mendes, "Supporting the Modeling of Flexible Educational Units. PoEML: a Separation of Concerns Approach," *J. UCS*, vol. 13, no. 7, pp. 980-990, 2007.

[4] Wil v. d. van der Aalst and Kees v. van Hee, Workflow Management: Models, Methods, and Systems (Cooperative Information Systems).: The MIT Press, 2002.

[5] Roberto Perez-Rodriguez, Manuel Caeiro-Rodriguez, and Luis Anido-Rifon, "Adding Process-Driven Collaboration Support in Moodle," in *Proceedings of 39th ASEE/IEEE Frontiers in Education Conference*, San Antonio, TX, 2009.

[6] Roberto Perez-Rodriguez, Manuel Caeiro-Rodriguez, and Luis Anido-Rifon, "Enabling Process-Based Collaboration Support in Moodle by Using Aspectual Services," in *Proceedings of 9th IEEE International*

Conference on Advanced Learning Technologies, Riga, Latvia, 2009, pp. 301-302.

[7] Olga C. Santos, Jesús G. Boticario, Elena del Campo, and Mar Saneiro, "IMS-LD as a workflow to provide personalized support for disabled students in Higher Education Institutions," in *User Modelling 2007*, Corfu, Greece, Corfu, Greece, 2007, pp. 41-49.

[8] D. Burgos, C. Tattersall, and R. Koper, "How to represent adaptation in eLearning with IMS Learning Design," *Interactive Learning Environments*, pp. 15(2), 161-170, 2007.

[9] P. Heinl et al., "A comprehensive approach to flexibility in workflow management systems.," in *Proc. Work Activities Coordination and Collaboration (WACC'99)*, San Francisco, CA, 1999, pp. 79-88.

[10] Manuel Caeiro, Contribuciones a los Lenguajes de Modelado Educativo. PhD. Vigo, Spain: Universidad de Vigo, 2007.

[11] Michael Adams, Arthur H.M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst, "Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows," in *Proc. 14th Int'l Conf. on Cooperative Information Systems (CoopIS'06)*, Montpellier, France, 2006, pp. 291-308.

[12] Barbara Weber, S. Rinderle, and M. Reichert, "Change Patterns and Change Support Features in Process-Aware Information Systems," in *Proc. 19th International Conference on Advanced Information Systems Engineering (CAiSE'07)*, Trondheim, Norway, 2007, pp. 574-588.

[13] T. Zarraonandia, C. Fernández, and J. M. Dodero, "A late modelling approach for the definion of computer-supported learning process," in *ADALE Workshop at Adaptive Hypermedia 2006*, Dublin, Ireland, 2006.

[14] P. McAndrew and M. Weller, "Applying Learning Design to Supported Open Learning," in *Learning Design*. Berlin: Springer-Verlag, ch. 17, pp. 281-290.

[15] M. Dougiamas and P. C. Taylor, "Moodle: Using learning communities to create an open source course management system," in *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, Hawaii, United States, 2003.

[16] Roberto Perez-Rodriguez, Manuel Caeiro-Rodriguez, and Luis Anido-Rifon, "Design of a Flexible and Adaptable LMS Engine in Conformance with PoEML," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 4, no. 0, 2009.

[17] Roberto Perez-Rodriguez, Manuel Caeiro-Rodriguez, and Luis Anido-Rifon, "Towards a PoEML-based Architecture for E-learning Systems," *IEEE RITA*, vol. 4, no. 3, pp. 230-238, August 2009.

[18] Joonsoo Bae, Hyerim Bae, Suk-Ho Kang, and Yeongho Kim, "Automatic Control of Workflow Processes Using ECA Rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 8, pp. 1010-1023, August 2004.

[19] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. (2003, June) SOAP Version 1.2 Part 1: Messaging Framework. [Online]. http://www.w3.org/TR/2003/REC-soap12-part1-20030624/