Journal of Universal Computer Science, vol. 16, no. 19 (2010), 2821-2840
submitted: 1/3/10, accepted: 29/9/10, appeared: 1/10/10 © *J.UCS*

Execution Model and Authoring Middleware Enabling Dynamic Adaptation in Educational Scenarios Scripted with PoEML

Roberto Perez-Rodriguez, Manuel Caeiro-Rodriguez
Luis Anido-Rifon, Martin Llamas-Nistal
(Telematics Engineering Department, University of Vigo, Spain
roberto.perez.rodriguez@gmail.com, mcaeiro@det.uvigo.es
lanido@det.uvigo.es, martin@det.uvigo.es)

Abstract: The design of adaptive e-learning systems has been approached from different points of view. Adaptive Educational Hypermedia (AEH) conceptual frameworks, usually decompose this problem into separate concerns: a User Model (UM), an Adaptation Model (AM), and a Domain Model (DM). Regarding Educational Modelling Languages (EMLs), they provide adaptation mechanisms such as the modelling of *participants* following conditional *learning paths* over a common *content structure*. The design of adaptive learning paths in EMLs (the Adaptation Model) is predefined during design-time, and no changes on it are allowed during run-time. In this paper we describe the support of dynamic adaptation features (run-time changes on models) using PoEML (Perspective-oriented EML) as modelling language, with focus on the execution model of the PoEML engine and on a SOA-based middleware used by authoring tools to invoke change primitives.
Key Words: Adaptive Learning Systems, Dynamic Adaptation, Educational Modelling Languages
Category: M.5

1 Introduction

Adaptation in web-based e-learning systems is being subject to intensive research nowadays. Many approaches to adaptation can be found out in the literature. Some of the approaches to the development of adaptive e-learning systems are implementation-oriented, that is, there is no high-level formalism to describe the behaviour of such systems. The other face of the coin is the model-oriented approach, which aims at capturing the requirements of the system in a model, being possible to develop several final systems compliant to the same model. Model-oriented approaches come principally from Adaptive Educational Hypermedia (AEH) and Educational Modelling Languages (EMLs) research fields [Hendrix et al. 2009].

In the AEH field, some works follow a separation-of-concerns approach and decompose the problem into layered models: a Domain Model (DM), an Adaptation Model (AM) and a User Model (UM)[1]. Regarding EMLs in general, and

[1] The LAOS framework [Cristea and de Mooij 2003] proposes a Goal and Constraints Model (GM) between the DM and the AM, in order to orientate the initial material.

IMS-LD [IMS 2003] in particular, they enable the modelling of Units of Learning (UoL) in accordance with many different pedagogical approaches, thus easing the reuse of such models [Koper 2002]. Differently from models that pivot around contents, an EML not only manages contents, but also activities to perform, involved roles, environments and available tools, the order in which activities have to be performed, etc. The modelling language PoEML[2] (Perspective-oriented Educational Modelling Language) [Caeiro-Rodriguez 2007][Caeiro-Rodriguez et al. 2007] is positioned in this conceptual field, and it constitutes the foundation for this work.

In many approaches to adaptation in e-learning accordingly to AEH or EMLs, models cannot be updated/modified in run-time. We term this approach as **static adaptation**, because they are based on preprocessing models to adapt them to users. The other approach is the dynamic one, by which some parts of the model may be updated/changed at run-time. We term this approach **dynamic adaptation**, classifying it in two levels, in function of the parts of the model that can be changed/updated in run-time:

- Level A. In this level, only user models allow run-time changes. The execution of the other models can vary in accordance with changes in the user model, but those models are not modified at all.

- Level B. In this level, it is allowed to change/update in run-time any part of the model, not just the user model. For example: the parts of the model that deal with the definition of activities to perform, order between activities, etc.

The support of this dynamic adaptive behaviour imposes several requirements[3] on the execution model of EML engines. Several research groups and international initiatives like the workshop on adaptation in CSCL'09 [Stahl and Kirschner 2009] have addressed this problem. However, suitable EML-based solutions to this problem have not been hitherto developed.

This paper introduces a solution to support dynamic adaptation in EML educational scenarios that also involves authoring functionality. In the one hand, we define an execution model flexible enough to enable run-time changes on the different perspectives of PoEML models. In the other hand, we present a SOA-based architecture whose central component is a PoEML execution engine developed in accordance with the PoEML execution model and that includes a run-time authoring interface based on change primitives.

[2] PoEML divides the modelling of adaptive courses into different perspectives following a separation-of-concerns approach.

[3] Constraints involve traceability (to conditionally select constructs from a running model) and on-the-fly reconfiguration of the model.

2 Related Work

Adaptive e-learning environments are the subject of some research projects, such as GRAPPLE [Hendrix et al. 2008], SALTIS[4] , and SUMA[5]. The design of adaptive e-learning environments has been approached from different research fields, mainly AEH (Adaptive Educational Hypermedia) and EMLs. We made an extensive literature review of existing proposals, classifying them as *static adaptation* or *dynamic adaptation*, and among the dynamic type we have checked at what extent it is allowed to modify the design at run-time, for example, if it is possible to make structural changes or only to change the value of some basic parameters.

LAOS [Cristea and de Mooij 2003] follows a separation-of-concerns approach to the design of AEH by dividing an integrated model into a Domain Model (DM), a Goals and Constraints Model (GM), a User Model (UM), an Adaptation Model (AM), and a Presentation Model (PM). The DM follows a hierarchical paradigm and the GM imposes completion dependencies and constraints between goals. This allows making explicit that information which otherwise would be implicit in the Adaptation Model and hidden in adaptation rules [Cristea and de Mooij 2003]. The AM is implemented by the LAG grammar [Cristea and Kinshuk 2003], which allows defining adaptive behaviour using a high-level language. In [Cristea and Kinshuk 2003] authors say that no run-time changes on the GM or AM are contemplated in their approach[6].

Regarding EMLs, IMS-LD does not impose a standard execution model [Vogten et al. 2006] and different approaches can be found in literature:

- In [Vidal et al. 2008], authors describe an execution model based on Petri nets[7]. This work does not address Petri net reconfiguration, which is a requisite to fully support dynamic adaptation.

- In [Vogten et al. 2006] an IMS-LD execution model based on Moore Machines is described: a User Model is comprised by a number of sets of Properties and they can be updated at run-time. This system is known as Coppercore engine, and there are no references to dynamic updating of the Adaptation Model, that is, reconfiguration of the Moore machines.

- In [Zarraonandia et al. 2006] the authors follow an Aspect-oriented approach to design an extension of the Coppercore engine. The authors' approach does not guarantee automatic consistency in the migration of a running UoL instance, therefore a further consistence check is needed.

[4] http://www.saltis.org/principles.html
[5] http://www.ines.org.es/suma/en/proyectos.php
[6] Authors say that not allowing to make run-time changes on the GM or AM makes sense because adaptation to one user should not modify the public domain.
[7] Traceability may be lost in the IMS-LD to Petri net translation.

Other e-learning specifications such as SCORM [Bohl et al. 2002] have also addressed adaptation. SCORM packages can be considered sequences of learning contents. In [Rey-López et al. 2006] it is proposed to extend SCORM at an activity level to enable conditional display of activities depending on the student's profile. In [Abdullah and Davis 2003] authors compare adaptive behaviour in Simple Sequencing and AEH.

In the literature, there are works based on the task/method paradigm. *Operationalisation languages* such as DSTM [Trichet and Tchounikine 1999] (that uses task/method based operationalisation primitives) translate *paper-based models* into *implemented models*. Two drawbacks on operationalisation languages are noted: (i) current operationalisation languages impose their modelling point-of-view[8], (ii) operationalisation languages do not propose any help for refining models. These two drawbacks are caused basically by the loss of traceability between the *paper-based model* and the *implemented model*. Regarding course-generation approaches, the one based on SCARCE [Tetchueng et al. 2008] composes courses before run-time, therefore it falls into the static adaptation type; in [Ullrich 2005] and [Marcke 1992], authors propose to postpone some adaptation decisions until run-time, so that they may take into account a dynamic UM (dynamic adaptation level A).

In summary, adaptation in e-learning has been usually approached as static adaptation or dynamic adaptation at level A. Some works extend e-learning specifications to support dynamic adaptation, but the scope of allowed changes does not cover all the elements in the EML in use. Our work differs from these in that we define an execution metamodel, enabling a complete dynamic replanning of the course, using a set of run-time change primitives. The requirements for this objective are explained in the next section.

3 Requirements

The life-cycle of an adaptive course is as follows. We are using *learning-by-doing* educational scenarios that follow Goal-Based Scenarios philosophy. A GBS *paper-based model* is then operationalised to a computer-understandable model by using PoEML. Finally, computer-understandable models are transformed to executable form and they are run by an execution engine.

To support dynamic adaptation some requirements are presented over the course life-cycle:

– To maintain structural correspondence between *paper-based* models and PoEML models, as well as between PoEML models and running models (traceability).

[8] They impose a particular definition of a task and a particular definition of how tasks are selected at run-time.

- To provide a full set of primitives for run-time changes on running-models. This entails to support on-the-fly reconfiguration of model structures.

Our objective is to support changes in running GBSs. The scope of changes is focused into *scenario operations*. In the literature, there are some approaches that use a pattern-based approach for classifying the non-predefined changes that are supported by the run-time environment, such as [Weber et al. 2007]. In particular, our approach was focused on the following change patterns: (i) insert task, (ii) delete task, (iii) move task, (iv) replace task, (v) swap task, (vi) extract sub task, (vii) inline sub task, (viii) embed task in loop, (ix) parallelise task, (x) embed task in conditional branch, (xi) add control dependency, (xii) remove control dependency, (xiii) update condition.

4 The Modelling Language

Following a separation-of-concerns approach, PoEML decomposes an integrated model into several *perspectives* (a perspective is a separated part of the modelling domain involving a specific purpose) and orthogonal concerns that we term *aspects* (different modes of control on the behaviour of each perspective), and that have to do with adaptation and personalisation. Perspectives involve issues that can be adapted, while aspects enclose the logic to decide what changes have to be performed. This separation of concerns approach is key for the reuse and adaptation of PoEML models. Perspectives are useful to selectively filter one part of the model.

The PoEML components are `elements`, `specifications`, and `expressions`. The assembly of language components allows designing a model in a "lego-like" way.

`Elements` represent things that may be part of an Educational Scenario (ES), which is the basic building block for creating PoEML models. An ES may include several types of elements, each one representing a particular purpose. The following elements are at disposal:

- `Causal Description` represents information that introduces and describes the ES.
- `Role` represents the participant that is going to carry out the tasks that are required in the ES.
- `Goal` represents the tasks that must be carried out in the ES, indicating what has to be done in it. An ES must include at least one Goal.
- `Environment` represents a space in which a task required in the ES has to be done. These spaces include the Tools that roles can use to achieve the Goals.

- `Tool` represents a functionality at disposal. Besides, PoEML also takes into account the explicit representation of the following components in Tools: Permissions, Operations, and Events.
- `Data Elements`. They allow representing the data used in the ES. ESs may include Variables, Goals may include Input and Output Parameters, Roles may include Attributes, Tools may include parameters in their Operations and Events, etc.

`Specifications` represent restrictions that may be set on the Elements of an ES. Two of the key specifications in an ES are:

- `Order` represents the Order in which the Sub-ESs of an ES have to be done.
- `Temporal` represents time at which the Sub-ES of an ES have to be started/finished.

`Expressions` represent change possibilities in accordance with the *aspects* that may affect to the structure or behaviour of the ES or its components.

- `Constant` represents a change possibility that is not going to change during run-time. They are important to facilitate making changes during design-time.
- `Condition` represents a change possibility in function of data inside a Data Element. Since Data Elements may be included in different Elements, these Expressions enable to define changes in Elements of a type in function of Elements from other types.
- `Signal` represents a change possibility in function of a signalling event that can be produced in an unexpected way.
- `Decision` represent a change possibility in function of the decision of a Role.

4.1 Declarative Modelling in PoEML

PoEML follows a declarative[9] (constraint-based) approach [Van der Aalst et al. 2009] to course modelling. The declarative approach gives a great freedom degree in run-time, because the execution control of a course is modelled as a set of constraints, and the consistency of a course instance is guaranteed whenever the constraints are not broken. In this sense, it can be said that in a course instance every state is permitted unless it is not explicitly forbidden. This approach provides a higher degree of flexibility than in traditional approaches, which use a procedural approach (step-by-step).

[9] A good approach that helps in run-time reconfiguration by easing a problem known as the *dynamic change bug* [Van der Aalst 2001].

Figure 1 shows a schematic representation of a course design inside the jPoEML graphical authoring tool. Boxes represent Scenarios, whilst circumferences represent Goals. It can be perceived how low-level Scenarios are aggregated into high-level ones. That composition relationship is represented in the figure by means of ESs including Sub-ESs. In order to support the modelling of relationships between Goals, two `elements` of the PoEML language are involved:

- `Attempt Relation` can be established between two Goals belonging to the same ES. In the Figure 1, an Attempt Relation is represented by a circle with a + symbol inside it. This indicates that to initiate Goal *Test Program* the Goal *Make UML Model* has to be satisfied first.

- `Completion Relation` can be set between a Goal in a ES and a Goal in one of its sub-ESs. In this sense, the consecution of a high-level Goal depends on the consecution of one or several low-level Goals. Completion Relations may be: (i) Completion AND, represented in Figure 1 as a circle with a & symbol inside it; (ii) Completion OR, represented in Figure 1 as a circle with a || symbol inside it; (iii) other types.

- `Goal Flow`. It serves to describe the direction of a dependency, the source is expressed by `input from` and the sink is expressed as an `output to`.

4.2 Formalisation of a Paper-Based Model with PoEML

PoEML is "agnostic" concerning pedagogy. This means that the PoEML metamodel is flexible enough to operationalise models from different pedagogical approaches.

We propose an example of a course in the context of a software engineering subject (see Figure 1). The *goal* of the course is twofold: to learn how to make UML models and to learn Java or C++ application programming. The *mission* is to make an UML model of a management application and later to develop it. The *cover story* presents a context for the practice, explaining the requirements of the software that has to be developed. *Feedback* is provided in the form of forums and chats. Finally, the *scenario operations* formalise the process of making the design as well as the software development process. *Scenario operations* are formalised as a network of PoEML Goals. The *role* of a student in this GBS course is twofold: analyst and programmer, whilst the *role* of a teacher is both senior analyst and project manager.

The operationalisation of this GBS consists *structurally* on a `root ES` that contains two Sub-ESs:

- The *Modelling* sub `ES` contains resources and tools for making UML models.

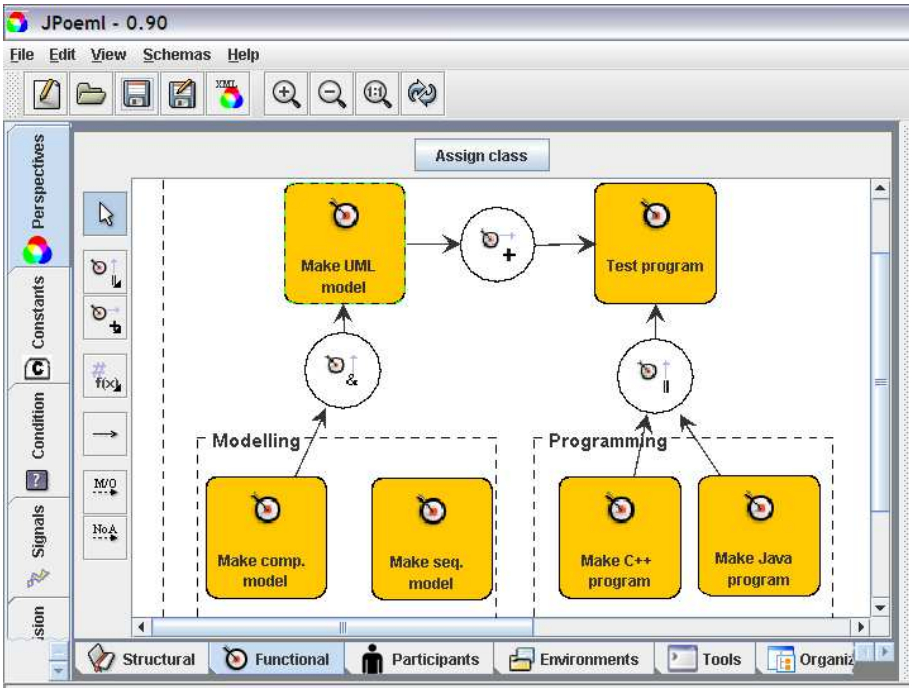

Figure 1: Stand-alone authoring with jPoEML.

– The *Programming* sub `ES` contains resources and tools for writing code in several programming languages, as well as for compiling and executing that code.

Functionally, the operationalisation of this ES consists on two `goals`:

– The *Make UML model* `goal` requires for its completion the previous completion of one sub `goal`: *Make component model*.

– The *Test program* `goal` requires for its completion the previous completion of one from two sub `goal`: *Make C++ program* or *Make Java program*. Additionally, an `output constraint` is attached to it, imposing a restriction on the percentile rate of test programs that the program must pass in order to considering it to be correct. This constraint is modelled by a `expression`.

5 Execution Metamodel

The execution metamodel enables PoEML ESs to be run in such a way that allows their behaviour to be modified before and during run-time. Object-oriented elements in the language are run as Finite State Machines (FSM). Modular FSM support this usage when designed following certain patterns, such as the hierarchical one [Lee et al. 2002] [Sklyarov 1999].

The exposition of the execution metamodel is threefold:

– A description of a *situation part*, that is, how the actual state of a FSM network is derived from the state of its contained FSMs (which maintain a structural correspondence to the elements in the modelling language).

– A description of a *execution part*, how a user-generated event triggers transitions on a FSM network.

– A description of a *change part*[10], that is, a change primitive triggers transition on a FSM network to move from situation "a" to situation "b".

5.1 The Situation Part: State Machines for ESs and Goals

A *situation* in the FSM network is an screenshot of the states of individual FSMs. We designed individual FSMs in such a way that it allows to evaluate its state when required. For that reason the figures of Scenario and Goal states do not include all the possible run-time transitions between states, instead they show the order that the evaluation procedure follows to check conditions and set the new state. Other PoEML elements are also instantiated as FSMs, but they are simpler that ES and Goal ones. A change in a FSM state may trigger the reevaluation of related FSMs.

5.1.1 ES Instances Life-Cycle

The execution states of an ES instance are shown in Figure 2. ES instance creation happens when its "parent" ES passes to state `Switched On`. It is an on-demand approach to the instantiation process: ES instances are created when they are needed, being thus a scalable approach. When the instance is created, it is at the `Not Accessible` state. Thus, the instance cannot be provided to be accessed by a participant. It is transition (`Not Accessible - Accessible`) the one that enables the instance to be accessed by a participant, and this transition is directly dependent on the Order and Temporal Perspectives (Order and Temporal constraints must be satisfied).

The other possible states are `Switched on`, `Switched off`, and `Locked`. An ES becomes `Switched on` when a participant enters it, and becomes `Switched off` when the last participant leaves it. An ES gets the `Locked` state when all the goals that it contains have been already achieved and some `Specification` restrains the access to it.

[10] The scope of allowed changes determines the level of dynamic adaptation.

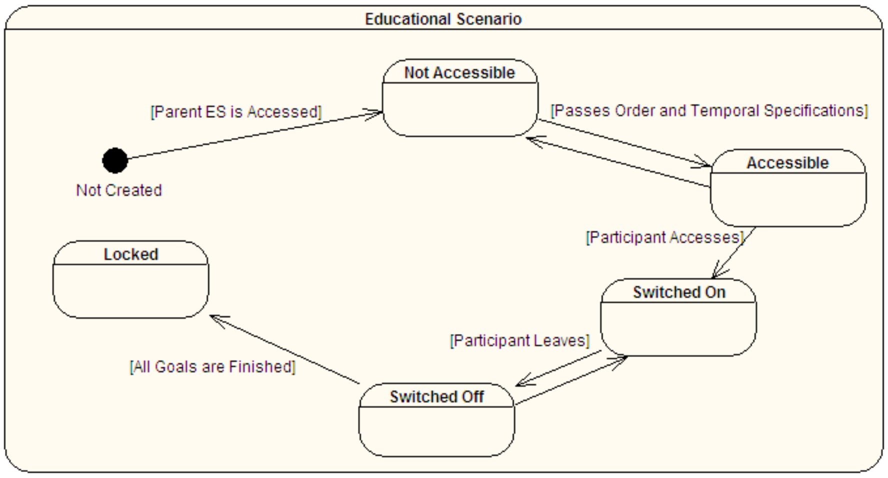

Figure 2: Execution states of an ES instance in PoEML

5.1.2 Goal Instances Life-Cycle

The execution states of a Goal are shown in Figure 3. The creation of a Goal instance happens when a "parent" Goal is attempted. This means that when a participant attempts a Goal, all the Goals that have `Completion dependencies` with it have to be instantiated. The Goal instantiation process is therefore very similar to the ES instantiation process, creating instances when they are needed. In a way similar to the ES instantiation process, a newly created Goal instance is not automatically ready to be attempted by a Participant. There are certain Attempt dependencies and Input Constraints that have to be satisfied in order to reach the `Attemptable` state.

When a participant attempts a Goal, its state is set to `Pending`. This means that somebody has attempted the Goal but its achievement has not yet been evaluated. Once it is evaluated, the Goal possible states are:

- `Failed`, when the Output Constraints and/or Completion Dependencies are not satisfied.
- `Achieved`, when the Output Constraints and Completion Dependencies are satisfied.
- `Expired`, when the deadline of the ES instance containing the Goal is over.

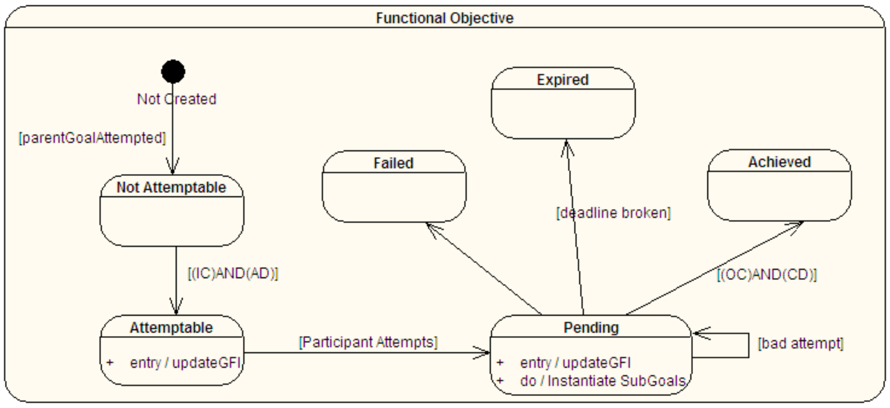

Figure 3: Execution states of a Goal instance in PoEML

5.2 The Execution Part

The execution part describes how a single change in the state of either a goal instance or an ES instance may trigger changes in related scenario and/or goal instances. The execution part also describes the strategy to propagate changes up in the course tree model. The execution model is based on Event-Condition-Action (ECA) rules. Table 1 shows a sample of them. When an event is produced, it may trigger some action whenever a certain condition is satisfied. In this sense, a single event may change the state of several elements in the hierarchy of goals and scenarios.

The two main requirements imposed over the execution metamodel are to guarantee termination and confluence. Figure 4 shows a network of FSMs that correspond to the objects in the PoEML design of Figure 1. Lines between objects represent the flow of triggered events between objects. As it can be seen in Figure 4, the events flow forms an Acyclic Graph, which guarantees that there are no loops in the design, so termination is assured. The events flow is of a hierarchical nature as well, as no concurrent events are contemplated, so confluence is also guaranteed. This is a quite conservative an very simple approach, but we consider it to be functional enough to design complex courses.

6 Dynamic Adaptation

In this section we describe the change part, that is, how a change in a course definition may trigger changes in its related course instances. On-the-fly reconfiguration is achieved automatically as all the changes trigger the re-evaluation of the states of all the related instances.

Associated to Element	Event	Condition	Action
ES	Update in the state of a ES	The new state is Switched On	Evaluate sub ESs
ES	Update in the state of a ES	The new state is Not Accessible	Evaluate sub ESs
Data Element	Update in the value of a Data Element		Evaluate Data Expressions that contain this Data Element
Data Expression	Update in the state of a Data Expression		Evaluate Goals that have this Data Expression as Input/Output Constraints
Goal Flow	Update in the state of a Goal Flow		Evaluate Goal Instances indicated by output
Goal	Update in the state of a Goal	The new state is Achieved	Evaluate Goal Flows that have this Goal as input
Goal	Update in the state of a Goal	The new state is Pending	Evaluate sub Goals
Goal	A Goal gets the Not Accessible state		Evaluate sub Goals; Evaluate Goal Flows that have this Goal as input

Table 1: Sample of ECA rules implementing the *execution part*.

Some change patterns introduced at the Requirements section can be implemented directly by one `change primitive`: *insert task* as `add goal`, *delete task* as `delete goal`, *replace task* as `update goal`, *Add control dependency* may be implemented as `add goal flow`, *update condition* as `update data expression`, etc. Others can be implemented as sequences of *change* `primitives`. For example, *move task* can be implemented as: (i) `remove goal flow`, (ii) `add goal flow`; *swap task* can be implemented as three `update attempt dependency` in sequence, etc.

Figure 5 illustrates this. It shows the same hierarchy of previous figure, but inside each object two states are drawn: the one in the left represents the initial state of the object (before the reevaluation), whilst the one in the right represents the state of the object after reevaluation. The analysed primitive is *add control dependency*. This transition is triggered by the addition of a dependency from the `goal` *Make Behavioural Model* to the `goal` *Make UML Model*. This change in the model causes the reevaluation of the goal hierarchy. This is a dual process:

– Events are propagated from the addition point to the root of the hierarchy. Elements change their state in cascade: CompletionDependency (False), MakeUMLModel (Pending), GoalFlow (False), AttemptDependency (False), TestProgram (Not Attemptable).

– Events are propagated from the addition point to leaves. The cascade of changes in elements is: GoalFlow (False), MakeBehaviouralModel (Attempt-

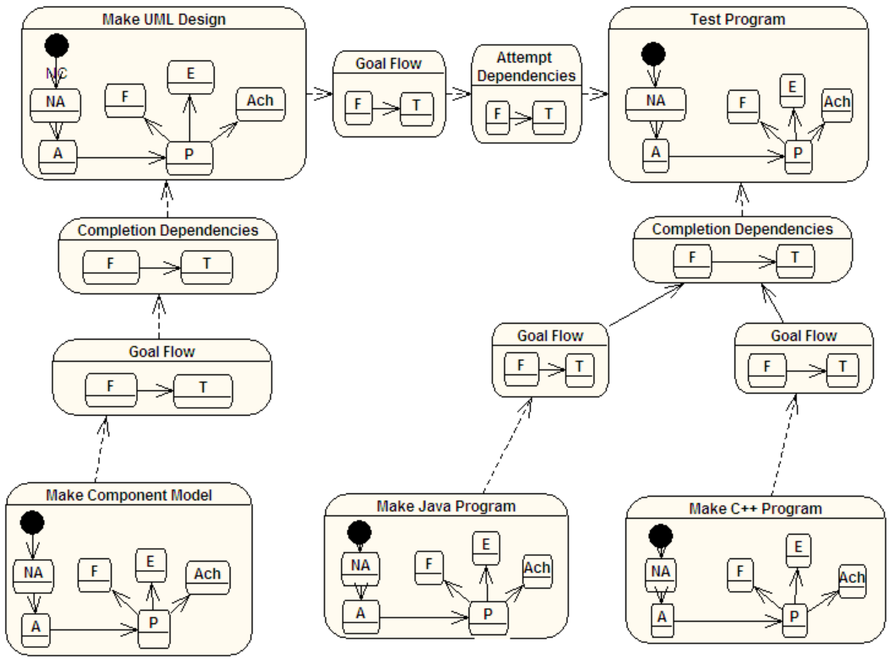

Figure 4: The FSM network with the PoEML design.

able).

7 Implementation

We have developed an implementation of an execution engine for PoEML courses in accordance with the execution model presented in this paper. The execution engine is integrated into the overall architecture of the e-learning system by a well-defined interface of Web Services. In this section we present the overall architecture based on the Service-orientation paradigm.

7.1 Overall Architecture

Figure 6 shows the overall architecture of the system, which is based on the service-orientation paradigm. The structure of a general e-learning system is composed of three layers: Presentation Layer, Business Logic Layer, and Database Layer. The purpose of this decomposition is to separate presentation-related concerns. As a consequence, over the same Business Logic Layer, it is possible to deliver courses with different presentation settings.

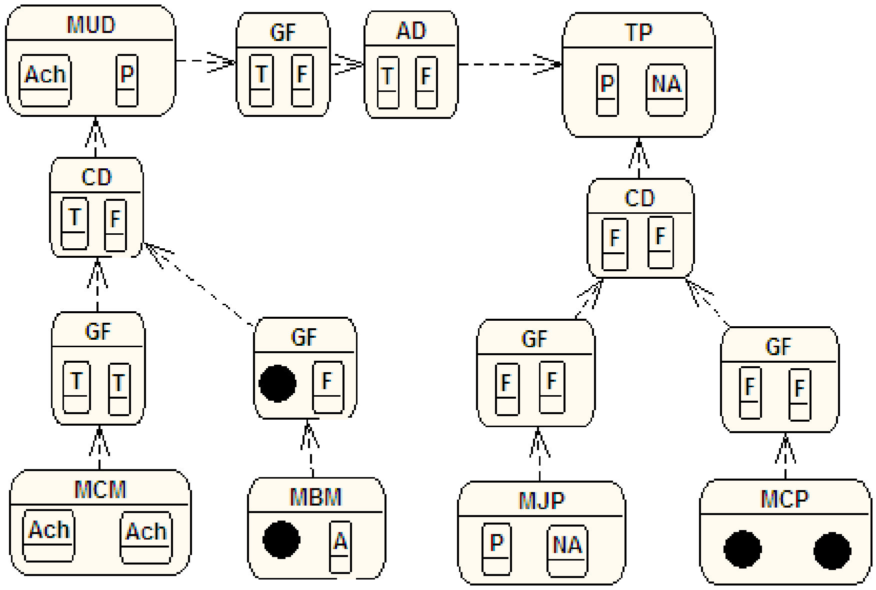

Figure 5: Initial and final state of the FSM network before and after adding a control dependency. Black dots represent the Not Created state.

The Presentation Layer is in charge of displaying educational scenarios, making use of the functionality provided by the Business Logic Layer. Presentation components are designed following a decomposition based on three main functionalities: authoring, monitoring and delivering. Table 2 lists the methods in the three interfaces provided by the Business Logic Layer. The Presentation layer has been developed in PHP[11], and uses the NuSOAP library[12] to consume service methods.

The Business Logic Layer has been implemented as a Web application running in Tomcat[13], and we use the Axis framework[14] to publish the service methods. The Business Logic Layer provides two main functionalities, namely: models management, and instances management:

– The **Models Manager** manages the educational scenario models. It maintains the version of models, and provides an authoring interface for updating them by an authorised user.

[11] http://www.php.net
[12] http://sourceforge.net/projects/nusoap
[13] http://tomcat.apache.org
[14] http://ws.apache.org/axis/

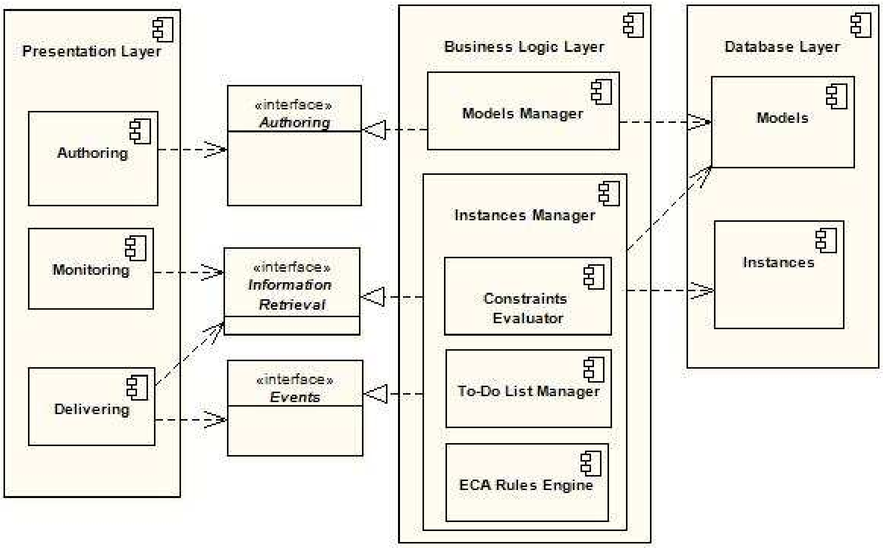

Figure 6: Overall architecture as a three-tier architecture.

– The **Instances Manager** is in charge of managing instances of PoEML elements. This component provides two interfaces: one for the passive retrieval of information, employed by the delivering component to display ESs; and the other one for the communication of events which are the result of participants' interaction with the presentation delivering component. This component is composed by three subcomponents: the ECA Rules Engine, which implements the *execution part*; the Constraints Evaluator, which implements the reevaluation of constraints after an update in the process model, or *change part*; and the To-Do List Extractor, which gets the pending tasks for each course instance from leaves to root in the hierarchy.

Finally, the Database Layer maintains two separate schemas: one for PoEML models, and another one for run-time instances. These two models are not fully independent, because a change in one scenario model may trigger as many changes in scenario instances as instances are running in the e-learning system. The Database Layer has been implemented in Oracle 11g.

7.2 Authoring

The authoring modes are two: not-connected mode, and connected mode.

– **Not-connected mode**. In Figure 1 we can see the same ES that we use throughout the paper as an example created with the jPoEML graphical

Method	Input Parameters	Output Parameters	Description
Information Retrieval			
getESInstancesByESId	id	:ESInstance[]	Retrieves all ESInstances from a ES
getESInstancesByParentId	id	:ESInstance[]	Retrieves all descendants from a ESInstance
getGoalInstancesByGoalId	GoalId	:GoalInstance[]	Retrieves all GoalInstances from a given GoalId
getEnvironmentInstances ByESInstanceId	ESInstanceId	:EnvironmentInstance[]	Retrieves all Environment Instances from a given ESInstance
getDataElementInstances ByESInstanceId	ESInstanceId	:DataElementInstance[]	Retrieves all DataElement Instances from a given ESInstance
Events			
updateDataElementInstance	id, name, description, type, ESId, ESInstanceId, value	id	Updates a DataElement Instance
setGoalInstanceOutput Parameter	id, outputParameter	id	Adds an OutputParameter to a GoalInstance
Authoring			
deployPoEMLManifest	manifest	ESRootId	Deploys a PoEML manifest
undeployPoEMLManifest	ESRootId	manifest	Undeploys a PoEML manifest
addGoalFlow	outputFrom, inputTo, goalConnector, value		Inserts a GoalFlow
getESsRoot		:ES[]	Retrieves all ESs that are root
getESsByParentId	parentId	:ES[]	Retrieves all ESs from its common ancestor
getGoalsByESId	id	:Goal[]	Retrieves all Goals in an ES
getDataElementsByESId	id	:DataElement[]	Retrieves all DataElements in an ES

Table 2: Methods in the execution engine interface. The authoring interface contains only a sample of get methods. There are similar methods for adding and updating elements. The : means that it is retrieved an object, and [] means that it is retrieved an array.

authoring tool. Once the design is done, jPoEML enables to export the design to XML. This XML manifest file can be imported in the execution engine by using the `deployPoEMLManifest` method in the authoring interface (see Table 2).

– **Connected mode**. In connected mode the user has to provide his credentials as well as the URL of the execution engine. In this authoring mode the authoring tool uses the capabilities of the database to make run-time changes. The author may invoke change primitives, and doing so the state of the model gets updated inside the authoring tool by invoking the get methods in the interface.

The representation of models inside the authoring tool is made in accordance to the PoEML separation-of-concerns approach. The authoring process can be separated in several steps/stages, and a different tab in the jPoEML environment is used in each step:

- In *stage 1* the knowledge domain is modeled as a set of `ESs` and `Sub ESs`. At this stage, the different elements that are part of ESs are aggregated: environments, goals, resources, tools, order and temporal specifications, etc.
- In *stage 2* the dependencies between `goals` are added to the model.
- In *stage 3* the different aspects are situated in the model.

These stages are overlapped for run-time changes, but having at our disposal a different tab in the authoring tool for representing each concern facilitates committing run-time changes and keeping track of them.

7.3 Delivering

The delivering component is in charge of displaying participants' working space. Typically, the working space provides some useful views, such as educational scenarios into which the participant is enrolled, and environments that are at the participant's disposal for communicating with other participants and for carrying out learning activities (see Figure 7).

The presentation layer calls the information retrieval interface to display ESs to a participant, calling for example the `getESInstances ByESId`, `getEnvironmentInstances ByESInstanceId`, `getDataElementInstances ByESInstanceId` and other related methods. When a participant finishes a learning activity, the presentation layer communicates to the business logic layer that the value of a Data Element (e.g. a grade in a quiz activity) has to be updated. This is done by calling the `updateDataElementInstance` method in the events interface.

8 Conclusions

In this paper we have presented an execution model and SOA-based middleware for courses scripted with PoEML that allows for run-time changes on course models, enabling on-the-fly reconfiguration of control structures. PoEML is very suitable for supporting on-the-fly changes due to two main characteristics:

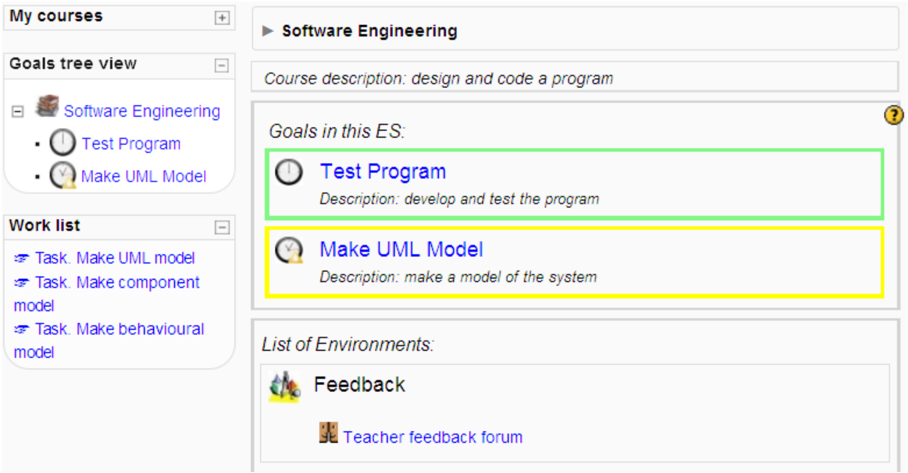

Figure 7: Screenshot of the course delivery.

– Its separation-of-concerns approach, that encapsulates different concerns in the modelling of courses as different elements in the language. This approach allows us to design a modular execution engine based on FSMs, and a modular interface for run-time changes, minimising crosscutting concerns.

– Its declarative approach, which allows a great flexibility, as everything is permitted unless it is explicitly forbidden by some constraint. This approach allows us to design a Constraints Evaluator, which re-evaluates all constraints in a course instance after a change has been committed on its related course model (and not only in the user model). This approach has shown to be more suitable than the step-by-step one, since the latter may lead to inconsistencies during the migration process.

Feedback from programmers that are developing different presentation modules permits us to list the following preliminary results/evaluation of our proposal:

– The division of the authoring interface in perspectives and aspects minimises the cognitive load of course authors.

– Our proposal allows a dynamic refining of PoEML models, because test users can be easily created to check the behaviour of refined models.

Acknowledgements

This work is co-funded by the European Community eContentPlus Programme ECP 2007 EDU 417008. The content of this paper is the sole responsibility of the authors. It does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of information contained here in.

References

[Abdullah and Davis 2003] Abdullah, N., Davis, H.: "Is Simple Sequencing Simple Adaptive Hypermedia?"; Proceedings of the Fourteenth ACM Conference on Hypertext and Hypermedia; 172–173; ACM, 2003.

[Bohl et al. 2002] Bohl, O., Schellhase, J., Sengler, R., Winand, U.: "The Sharable Content Object Reference Model (SCORM)–A Critical Review"; Proceedings of the International Conference on Computers in Education; 950–951; Citeseer, 2002.

[Caeiro-Rodriguez 2007] Caeiro-Rodriguez, M.: Handbook of Visual Languages for Instructional Design: Theories and Practices; chapter PoEML: A Separation-of-Concerns Proposal to Instructional Design, 185–209; Information Science Reference, 2007.

[Caeiro-Rodriguez et al. 2007] Caeiro-Rodriguez, M., Marcelino, M. J., Llamas-Nistal, M., Anido-Rifon, L., Mendes, A. J.: "Supporting the Modeling of Flexible Educational Units PoEML: A Separation of Concerns Approach"; Journal of Universal Computer Science; 13 (2007), 7, 980–990.

[Cristea and de Mooij 2003] Cristea, A., de Mooij, A.: "LAOS: Layered WWW AHS Authoring Model and their Corresponding Algebraic Operators"; WWW03 (The Twelfth International World Wide Web Conference), Alternate Track on Education, Budapest, Hungary; Citeseer, 2003.

[Cristea and Kinshuk 2003] Cristea, A., Kinshuk, K.: "Considerations on LAOS, LAG and their Integration in MOT"; (2003).

[Hendrix et al. 2009] Hendrix, M., Cristea, A., Burgos, D.: "Comparative Analysis of Adaptation in Adaptive Educational Hypermedia and IMS-Learning Design"; (2009).

[Hendrix et al. 2008] Hendrix, M., De Bra, P., Pechenizkiy, M., Smits, D., Cristea, A.: "Defining Adaptation in a Generic Multi Layer Model: CAM: The GRAPPLE Conceptual Adaptation Model"; Proceedings of the 3rd European Conference on Technology Enhanced Learning; Springer, 2008.

[IMS 2003] IMS, L.: "Learning Design specification v1"; Retrieved: November; 20 (2003), 2004.

[Koper 2002] Koper, R.: "Modeling Units of Study from a Pedagogical Perspective-The Pedagogical Metamodel behind EML,(2001)"; Educational Expertise Technology Centre, Open University of the Netherlands; (2002).

[Lee et al. 2002] Lee, S., Yoo, S., Choi, K.: "Reconfigurable SoC Design with Hierarchical FSM and Synchronous Dataflow Model"; Proceedings of the Tenth International Symposium on Hardware/Software Codesign; 204; ACM, 2002.

[Marcke 1992] Marcke, K.: Instructional Models in Computer-Based Learning Environments; chapter A Generic Task Model for Instruction, 171–194; Springer-Verlag, 1992.

[Rey-López et al. 2006] Rey-López, M., Fernández-Vilas, A., Díaz-Redondo, R., Pazos-Arias, J., Bermejo-Muñoz, J.: "Extending SCORM to Create Adaptive Courses"; Lecture Notes in Computer Science; 4227 (2006), 679.

[Sklyarov 1999] Sklyarov, V.: "Hierarchical Finite-State Machines and their Use for Digital Control"; IEEE Transactions on Very Large Scale Integration (VLSI) Systems; 7 (1999), 2, 222–228.

[Stahl and Kirschner 2009] Stahl, G., Kirschner, P.: "Introduction to CSCL 2009 Workshops, Tutorials and Seminars"; Proceedings of the 9th International Conference on Computer Supported Collaborative Learning-Volume 2; 207; International Society of the Learning Sciences, 2009.

[Tetchueng et al. 2008] Tetchueng, J., Garlatti, S., Laube, S.: "A Context-Aware Learning System Based on Generic Scenarios and the Theory in Didactic Anthropology of Knowledge"; International Journal of Computer Science and Applications; 5 (2008), 71–87.

[Trichet and Tchounikine 1999] Trichet, F., Tchounikine, P.: "DSTM: a Framework to Operationalise and Refine a Problem Solving Method Modeled in Terms of Tasks and Methods"; Expert Systems With Applications; 16 (1999), 105–120.

[Ullrich 2005] Ullrich, C.: "Course Generation Based on HTN Planning"; Proceedings of 13th Annual Workshop of the SIG Adaptivity and User Modeling in Interactive Systems; 74–79; Citeseer, 2005.

[Van der Aalst 2001] Van der Aalst, W.: "Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change"; Information Systems Frontiers; 3 (2001), 3, 297–317.

[Van der Aalst et al. 2009] Van der Aalst, W., Pesic, M., Schonenberg, H.: "Declarative Workflows: Balancing Between Flexibility and Support"; Computer Science-Research and Development; 23 (2009), 2, 99–113.

[Vidal et al. 2008] Vidal, J., Lama, M., Sanchez, E., Bugarın, A.: "Application of Petri Nets on the Execution of IMS Learning Design Documents"; Proceedings of EC-TEL 2008; 461; Springer-Verlag New York Inc, 2008.

[Vogten et al. 2006] Vogten, H., Tattersall, C., Koper, R., Van Rosmalen, P., Brouns, F., Sloep, P., van Bruggen, J., Martens, H.: "Designing a Learning Design Engine as a Collection of Finite State Machines"; International Journal on E-Learning; 5 (2006), 4, 641.

[Weber et al. 2007] Weber, B., Rinderle, S., Reichert, M.: "Change Patterns and Change Support Features in Process-Aware Information Systems"; Lecture Notes in Computer Science; 4495 (2007), 574.

[Zarraonandia et al. 2006] Zarraonandia, T., Dodero, J., Fernández, C.: "Crosscutting Runtime Adaptations of LD Execution"; Journal of Educational Technology and Society; 9 (2006), 1, 123.