An Execution Semantics to Support Flexibility in Collaborative Learning Scripts

Roberto Pérez-Rodríguez, Manuel Caeiro-Rodríguez, University of Vigo, Spain
Email: rperez@gist.det.uvigo.es, Manuel.Caeiro@det.uvigo.es

Abstract: Flexibility is a key concern on the adoption of model-based solutions to computationally support collaborative learning. These solutions address the structuring of collaborative activities in pre-specified ways, maximizing the chance of occurrence for certain desirable interactions. Nevertheless, it is very difficult, if not impossible, to be able to predict the unfoldment of collaborative learning activities. Therefore, a flexible solution that allows changes during the run-time is a basic requirement. This paper presents a solution to this problem by following an object-oriented modelling approach. It also involves an execution semantics based on hierarchical finite state machines networks. Together, these two approaches enable to manage the enactment of collaborative learning models and to perform changes in a non-disruptive way. The paper uses a typical JIGSAW activity to illustrate the need for this kind of solution.

Introduction

Model-based solutions have been widely discussed in the field of Computer Supported Collaborative Learning (CSCL). There have been several model-based initiatives with different names, mainly CSCL scripts (Dillenbourg, 2002; Dillenbourg & Hong, 2008), but also Educational Modelling Languages (Rawlings et al. 2002) or Visual Instructional Design Languages (Botturi & Stubbs, 2007; Caeiro-Rodríguez et al., 2010), such as the IMS Learning Design (IMS LD) specification (IMS, 2003). The key idea underlying these initiatives is to provide a computational modelling language that enables the description of collaborative learning plans. Later, such descriptions can be processed by appropriate computer applications in order to enact the corresponding collaborative learning experiences. In this way, the application can arrange a particular and precise learning environment for each user (eg. learner, teacher) and activity. A variety of issues can be managed in order to structure the learning plan (Kollar et al., 2006), such as the activities that should be carried out by each participant, how participants are sorted into groups, the order in which activities should/could be performed, the transfer of resources among activities, the environments in where each participant should work, etc. These functionalities facilitate the management and development of collaborative activities, providing support both for teachers and learners by triggering group interactions in certain desirable ways. Eventually, the way in which the computer application can support collaborative learning depends on the expressiveness of the computational modelling language. Nevertheless, nowadays expressiveness is not a main concern in these model-based solutions.

Flexibility has become the main concern towards the adoption of model-based solutions to support collaborative learning (Dillenbourg & Tchounikine, 2008). Some times authors have argued against the development of this kind of solutions because collaborative learning is highly unpredictable and it cannot be prescribed in a lesson plan (Dillenbourg, 2002). Many times, plans need to be modified and changed continuously during the enactment (run-time). A solution to this problem is adaptation (Ferraris et al. 2009; Specht & Burgos 2006), taking into account during the design-time the various alternatives or situations that can be produced later. In this way, some approaches support some kind of pre-defined adaptations that enable to capture the possible alternatives and the conditions to choose among them during run-time. More over, other solutions also include late-modelling constructs that enable to delay the modelling of certain parts to the future, after the learning process has already begun (Zarraonandia et al., 2006). These solutions provide some degree of flexibility to model-based solutions, but it is not enough. A complete solution should enable to perform any change to the learning plan at any time during its enactment, without bringing the system down.

This article shows a solution to support flexibility during the enactment of learning plans. This proposal is based on the adoption of an object-oriented modelling solution inspired on the workflow domain (Redding et al., 2010) instead the more typical task oriented approaches (as in IMS LD). Three of the key mechanisms involved in this proposal are: (i) the management of instances for each element involved in the enactment of the learning plan maintaining a particular state for each instance; (ii) the definition of an execution semantics based on event-condition-actions (ECA) rules that define the behaviour of the different elements and their relations; and (iii) the definition of an API that enables to monitor the state of the different instances, to make changes on the model or to force the state of certain instances. This solution enables to perform changes in the models without bringing the supporting application down (namely: hot changes).

Flexibility

The description of collaborative learning plans expressed as computational models involves a intrinsic dilemma (Dillenbourg & Tchounikine, 2008): "*if the scaffolding is too weak, it will not produce the expected interactions; if it is too strong, it will spoil the natural richness of collaboration*". In addition, as argued in the previous introduction, it is really difficult if not impossible to predict and gather all possible alternatives in learning plans. The solution to this problem is flexibility, defined as the capacity of a computational application that enacts learning plans to change its behaviour in not previewed ways when requested by an authorized user (i.e., a teacher or a learner). In other words, learning plans prescribe certain behaviours for the supporting application in terms of several issues: assignment of activities to users; grouping of users; transfer of resources; composition of learning environments, etc. During the enactment learning plans may need to be changed and the system has to behave appropriately supporting such changes without requiring to re-start the system.

Let see a typical collaborative learning use case where flexibility is required. We consider a common collaborative scenario: the JIGSAW. JIGSAW is a well-known technique for collaborative learning (Aronson et al., 1978). Students typically use it in face-to-face settings without computer support (see http://www.jigsaw.org). The typical JIGSAW technique involves three stages as follows: (i) group formation and individual reading; (ii) expert groups; (iii) collaborative groups.

Several authors have developed computer applications supporting the management of JIGSAW learning activities (Gallardo et al., 2003). Moreover, a main issue in some of these developments has been the provision of adaptations and flexibility support (Pérez-Sanagustín et al., 2009). The support of the JIGSAW by a computer application demands three types of flexibility solutions:

- Type 1: **expected variations**. This is the basic typical requirement as identified in Pérez-Sanagustín et al. (2009). In the JIGSAW case we may have variations in relation with the total number of learners, the number of parts in which the lesson materials are divided, etc. In addition, we could consider if these variations can be produced just before the system begins to enact the JIGSAW (configuration time), or if they can be produced once the enactment has already begun. The first situation involves a typical adaptation problem, whereas the second demands a more flexible solution as it is required to change the execution state of the application.
- Type 2: **global variations**. These are changes that affect to the whole learning plan and therefore they have to be translated to all the instances during the enactment. For example, to include a new task between the first stage and second JIGSAW stages in order to clarify the general goal of the activity to all learners. This type of change affects to all the participants involved in the system. Similarly to the previous type, this change can be produced when no instance of the second stage has been initiated (which could be considered as an adaptation problem), or when some of those instances have already been launched. This last option is more complex as we need to revert the system to a previous stage, where learners cannot continue working in stage 2 until the new task is concluded.
- Type 3: **particular variations**. These are changes that affect to specific elements during the enactment. For example, if a learner drops-out some activities, they are not going to be concluded or some groups will not be able to work in the intended way. Therefore, the enactment system may need to change the state of some instances in order to continue supporting the overall learning process. These changes will affect just to particular instances and not to the whole enactment.

Following sections introduce our solution for supporting these three types of flexibility. It is mainly provided through the execution semantics of the supporting application. In addition, the execution semantics is built on top of PoEML, acronym of Perspective-oriented Educational Modelling Language (Caeiro-Rodríguez et al., 2007).

Execution Metamodel

PoEML follows a declarative (constraint-based) approach to the modelling of learning plans (van der Aalst et al., 2009). The declarative approach gives a great freedom degree in run-time, because the execution control of a lesson plan is modelled as a set of constraints, and the consistency of instances is guaranteed whenever the constraints are not broken. The execution metamodel describes the enactment of the computational PoEML models. In our case, this metamodel follows an object-oriented approach and it is developed in the following ways:

- The elements in the modelling language are *classes* and their instances are *objects*. PoEML includes elements used in the specifications and expressions that are also translated into *classes* and then *objects* in the execution model. These elements are used to establish relationships among the main PoEML elements.
- The set of possible execution states for an *object* is modelled as a Finite State Machine (FSM).
- A complete *execution model* is composed by a hierarchical network of interrelated Finite State Machines.

This hierarchical FSM approach facilitates the introduction of changes during enactment (Lee et al. 2002). In order to support the maximum degree of flexibility, the following has to be noted:

- A *declarative approach* is key to overcoming the so called *dynamic change bug*, that is, possible inconsistencies in the *model instance* migration process after a change (van der Aalst et al., 2009).
- Traceability between *designed (paper-based) models*, *computational (PoEML) models*, and *executing models* is key to supporting run-time changes. In other solutions there are irreversible translation from the *designed* to *computational* and to *executing* models that makes impossible to trace back the execution.

Another main issue in this metamodel is the instances creation process. Instance creation is produced in an on-demand approach. Each element instance is created when it is needed and this happens when its parent container (typically an ES) is accessed by an user. This is a very scalable solution that facilitates the performance of changes in those parts of the models that have not been instantiated yet (late modelling). In other way, instance deletion is not possible. Instead, *cancelled* and *orphan* states are identified to represent the deletion of a certain element. In this way it is facilitated the update of the related elements' FSM.

Formalisation

The execution metamodel is defined as three separated parts for the sake of clarity:

- The **situation part** defines the FSMs for all the *classes* in PoEML.
- The **execution part** defines how an event in one FSM changes the state of the FSM network in which it is contained.
- The **change part** defines how a change in the *model* changes its related *model instances* (FSM networks).

Situation Part

The **situation part** of a FSM network is a screenshot of the individual FSMs that it contains. Individual FSMs are designed in such a way that it allows on-demand re-evaluation, that is, the state of a particular *object* can be re-calculated whenever it may be needed, because it depends only on the value of its contained *properties*. As PoEML models consists of networks of related elements, changes are propagated through FSMs in an automated way.

Next figures show two of main FSMs of the PoEML execution metamodel: ES and Goal. In the top left part of these figures is compiled the set of properties of these classes. Changes in these classes determine the transition from one state to another. It is aslso represented the methods performed when each state is achieved. For example, in Figure 1 when an ES is cancelled its contained Goal instances have to be cancelled too.

Figure 1 shows the extended state-transition diagram for an ES. The possible states are:

- **Not created**: the ES has not been created yet.
- **Not accessible**: the ES has been created but participants cannot access to it yet. The possible reasons may be that other ESs have to be done beforehand, or that the ES is scheduled to be accessed at a particular time.
- **Accessible**: participants are able to access the ES.
- **Active**: at least a participant has accessed the ES.
- **Finished**: there are no remaining goals to be achieved in the ES.
- **Orphan**: the parent ES is cancelled or orphan.
- **Cancelled**: the ES has been cancelled by an authorised user.

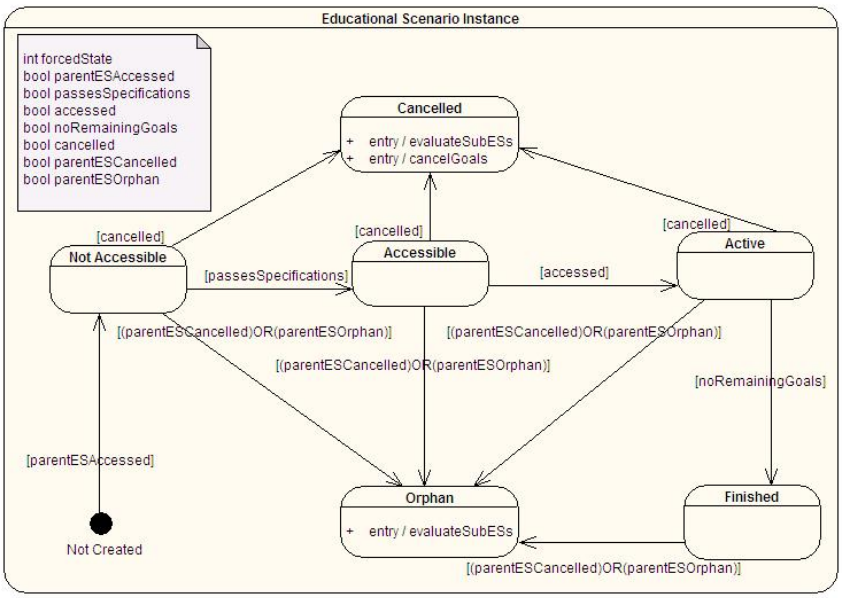

Figure 1. ES Instance Finite State Machine.

In a way similar to ESs, Goals have also an associated state-transition diagram. The possible states are:

- **Not created**: the Goal has not been created yet.
- **Not accessible**: the Goal has been created but participants cannot attempt it yet. The possible reasons may be that there are whether an unsatisfied input constraint or attempt dependency.
- **Attemptable**: participants are able to attempt the Goal.
- **Pending**: the Goal has been attempted, but its achievement has not been evaluated yet.
- **Failed**: participants have failed to achieve the Goal.
- **Achieved**: participants have succeeded to achieve the Goal.
- **Expired**: the temporal deadline has been reached.
- **Orphan**: the parent Goal is cancelled or orphan.
- **Cancelled**: the Goal has been cancelled by an authorised user.

Execution Part

The **execution part** deals with the strategy to propagate events in a FSM network. It clearly describes how a single update in the state of an *object* triggers changes up in the *model instance* into which it is contained, updating the state of related *objects*. The two key requirements are to support termination and confluence. Termination means that the chain of triggered events must not continue indefinitely. Confluence means that in for the same set of initial *situations* and triggered events, the final situation after two different executions must be the same one. These two requirements are supported as follows:

- The events flow is always an acyclic graph, guaranteeing that there are no loops, so termination is guaranteed
- Also, the events flow is hierarchical, and there are no concurrent events, so execution is always confluent.

Change Part

The **change part** deals with run-time changes in *models* and *model instances*. In particular, it manages:

- Changes in *models*: these are a set of primitives that enable to commit atomic changes on running *models*. Here, two strategies are considered:
  - To create a new version of the *model*. With this strategy, all future *model instances* will conform to a new version of the *model*, whilst old *model instances* will still be following an old version of the *model*. This approach is conservative, but it is useful in some use cases.
  - To update all *model instances*. With this strategy, all *model instances* are migrated to a new version of the *model*.
- Changes in *model instances*: these are a set of primitives that enable to commit atomic changes on running *model instances* without affecting other running *instances* from the same *model*.

On-the-fly reconfiguration is achieved automatically as all the changes trigger the re-evaluation of the states of all the related instances.

Architecture & Implementation

We have developed an implementation of an execution engine for PoEML courses that conforms to the execution model presented in this paper and that follows a typical three-layer design pattern. The *presentation layer* of the system has been developed in PHP and uses the NuSOAP library to consume the service methods of the execution engine. The execution engine forms the *business logic layer*. It has been implemented as a Web application running in Tomcat and we use the Axis framework for publishing the service methods. Table 1 gathers some of the methods provided by the execution engine. Finally, there is a *database layer* that 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 element of the model may trigger as many changes in the corresponding instances as instances are running in the execution engine. The Database Layer has been implemented in Oracle 11g.

Conclusions

CSCL is devoted to search for environments that directly or indirectly favour the emergence or rich and intensive interactions. Model-based approaches try to facilitate and promote the performance of certain desirable interactions by structuring the computational support provided by appropriate applications. Nevertheless, fixing such structuring and the corresponding degree of coercion is a delicate design choice (Dillenbourg & Tchounikine, 2008): "*too constrained environments would spoil the richness of collaborative interactions; too open environments would fail to induce the desired collaborative interactions*". A solution to this problem is achieved by providing flexibility in supporting applications. The computer application should behave as specified in the modelled plan as long as it unfolds in the desirable way. Nevertheless, if some difficulty is detected or some unexpected concern appears, the system should allow to modify the initial plan and to continue

supporting the learning process without disrupting it. The solution presented is a step toward this based on an object-oriented modelling approach and on hierarchical finite state machines networks execution semantics.

Table 1: Sample of service methods in the execution engine interface. The ':' means that it is retrieved an object, and [] means that it is retrieved an array.

Method	Input parameters	Output parameters	Description
Information retrieval			
getESInstancebyESId	id	:ESInstance[]	Retrieves all ES Instances from a ES
getEnvironmentInstancesByESInstanceId	ESInstanceId	:EnvironmentInstance[]	Retrieves all Environment Instances from a given ESInstance
Authoring/changes			
setESMultiplicity	id, multiplicity		Inserts the multiplicity of an ES
cancelGoalInstance	id		Cancels a Goal instance
updateDeadline	ESId, deadline		Changes a deadline

References

Aronson, E., Blaney, N., Sikes, J., Stephan, G. & Snapp, M. (1978). The jigsaw classroom. CA: Sage.

Botturi, L., Stubbs, T. (eds.) (2007). *Handbook of Visual Langauges in Instructional Design: Theories and Pratices. Hershey*, PA: Idea Group.

Caeiro-Rodríguez, M., Marcelino, M. J., Llamas-Nistal, M., Anido-Rifón, L. & Mendes, A. J. (2007). Supporting the modelling of flexible educational units PoEML: a separation of concerns approach. *Journal of Universal Computer Sciences, 13*(7), 980-990.

Caeiro-Rodríguez, M., Derntl, M. & Botturi, L. (2010). Special issue on visual instructional design languages. *Journal of Visual Languages and Computing, 21*(6), 311-312.

Dillenbourg, P. (2002). Over-scripting CSCL: The risks of blending collaborative learning with instructional design. In P. A. Kirschner (Ed.), *Three worlds of CSCL. Can we support CSCL* (pp. 61-91). Heerlen.

Dillenbourg, P. & Tchounikine, P. (2007). Flexibility in macro-scripts for computer-supported collaborative learning. *Journal of Computer Assisted Learning, 23*(1), 1-13.

Dillenbourg, P. & Hong, F. (2008). The mechanics of CSCL macro scripts in Computer-Supported Collaborative Learning 3:5-23.

Ferraris, C., Vignollet, L., Martel, C., Harrer, A., & Dimitriadis, Y. (2009). Competitive challenge on adapting activities modeled by CSCL scripts. Workshop at the CSCL 2009. June 8, 2009, Rhodes, Greece.

Gallardo, T., Guerrero, L. A., Collazos, C., Pino, J. A. & Ochoa, S. (2003). Supporting JIGSAW-type collaborative learning. Proceedings of the 36th Annual Hawaii International Conference on System

IMS (2003). IMS Learning Design: Information Model, Best Practice Guide, Information Binding, XML examples and Schemas.

Kollar, I., Fischer, F., & Hesse, F. (2006). Collaboration Scripts--A Conceptual Analysis. Educational Psychology Review, 18(2), 159-185.

Lee, S., Yoo, S. & Choi, K. (2002). Reconfigurable SoC design with hierarchical FSM and synchronous data flow model. Proceedings of the tenth International Symposium on Hardware/Software Codesign.

Pérez-Sanagustín, M., Burgos, J., Hernández-Leo, D., & Blat, J. (2009). Considering the intrinsic constraints for groups management of TAPPS & Jigsaw CLFPs. Proceedings of International Conference on Intelligent Networking and Collaborative Systems (INCoS 2009). November, 2009, Barcelona, Spain.

Rawlings, A., van Rosmalen, P., Koper, R, Rodríguez-Artacho, M. & Lefrere, P. (2002). *Report on Educational Modeling Languages*. CEN Workshop Aggreement.

Redding, G., Dumas, M., ter Hofstede, A. H. M. & Iordachescu, A. (2010). A Flexible, Object-centric Approach for Business Process Modelling. *Service Oriented Computing and Applications, 4*(3):191-201.

Specht, M. & Burgos, D. (2006). Implementing adaptive educational methods with IMS Learning Design. Proceedings of Adaptive Hipermedia (AH 2006). June, 2006, Dublin, Ireland.

van Der Aalst, W. M. P., Pesic, M., & Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support, *Computer Science Research and Development, 23*(2), 99-113. Springer.

Zarraonandia, T., Fernández, C., & Dodero, J. M. (2006). A Late Modelling Approach for the Definition of Computer-Supported Learning Process. Proceedings of Adaptive Hypermedia. (AH 2006). June, 2006, Dublin, Ireland.

Acknowledgments

We want to thank *Ministerio de Ciencia e Innovacion* for its support to this work under grant "Methodologies, Architectures and Standards for adaptive and accessible e-learning (Adapt2Learn)" (TIN2010-21735-C02-01).