CSCL scripts: interoperating table and graph representations

Péricles Sobreira, Pierre Tchounikine, Université Joseph Fourier, Grenoble, France
Email: Pericles.Sobreira@imag.fr, Pierre.Tchounikine@imag.fr

Abstract: This article shows how teachers may be offered complementary means in the form of a table representation (featuring simplicity and intuitiveness) and workflow-based representations (featuring data/work flow and scheduling).

Representing macro-scripts: tree-, workflow- and table-based approaches

CSCL macro-scripts are coarse-grained pedagogy-oriented scenarios which aim to set up conditions (guidance and constraints) to improve the likelihood that knowledge-generating interactions such as explanations and engaging in argumentation, negotiation, conflict resolution, or mutual regulation occur (Dillenbourg & Tchounikine, 2007). An example is the jigsaw script with the following pattern: first, participants individually work on a topic; second, students having worked on the same topic meet in "expert groups" to exchange ideas; third, "jigsaw groups" are formed by grouping students who each worked on a different topic in the preceding phase; finally, all students join for a debriefing session. Macro-script editing includes reflecting on the way groups are created, refining the breakdown of a task in relation to past or future classroom activities, or moving a resource from one activity to another. Script management may also require run-time adaptations. Teachers must thus be empowered to reflect on the script and adapt it before and during its enactment. This is an issue as it requires representations means to be intuitive and easy to use and, at the same time, operational and/or interoperable with complementary operational means.

Most scripting languages/editors adopt a semi-formal syntax based on trees or graphs. Tree-based representations may be found in many projects whose representational perspective is based on or inspired by XML, e.g., Reload (Reload html). Graph-based representations based on or inspired by workflow or automata representations are rather dominant; see (Botturi & Stubbs, 2007) for a review. The main advantage is that graph representations neatly capture the dynamics (dataflow and/or workflow) which is present in many scripts, and allows a straightforward operationalization with workflow engines (Harrer et al., 2009). However, although these languages/editors may be considered intuitive thanks to their graphical syntax, some works highlighted that using representations inspired from data or process modeling such as XML-like trees or process charts, which are not widespread among teachers, may be an issue for adoption (Neumann et al., 2010).

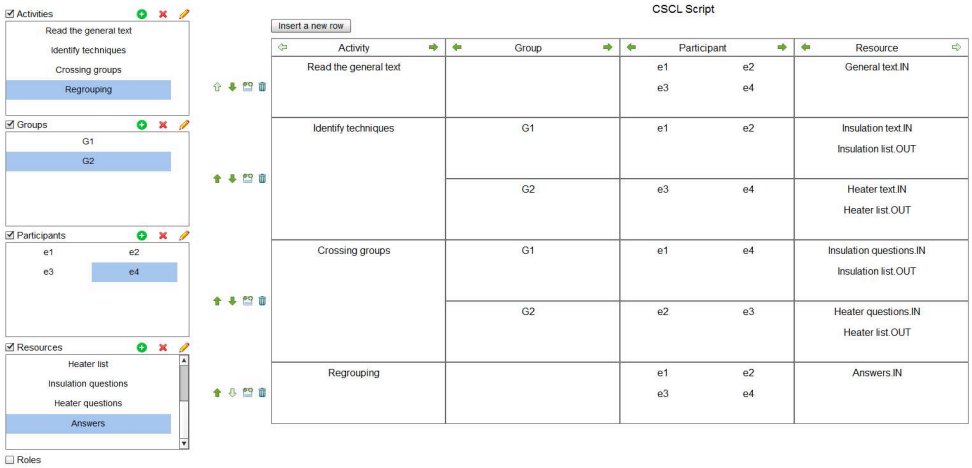

Figure 1. A script as a table, using the ediT2 editor.

An alternative approach is to offer representation means based on a structure which is known to be within teachers' basic ICT skills. This is the approach adopted by the ediT2 editor (Sobreira & Tchounikine, 2012), which represents macro-scripts as tables. Figure 1 presents its general interface, which is similar to that of a table editor in an office suite. Columns represent the activities, groups, participants, resources, and roles, notions that have been identified within a consensus effort for representing CSCL scripts (Kobbe et al., 2007). A row corresponds to a task, and can be broken down into sub-rows using splitting and merging facilities. Items (i.e. particular activities, participants, or roles) are created in the left part of the interface, and can then be dragged and dropped in the table cells. Different facilities are provided such as moving a row up or down, removing or duplicating a row (with or without its items), moving or copy-and-pasting items from one cell to another. Thanks to the table structure, a specific feature of this editor is that it does not impose a predefined representation structure: the different notions (activities, participants, etc.; or, in other words, the different columns) can be put in the order one wants, this order still being modifiable while the table is partially filled.

Teachers may thus contextually decide to use a representation pattern such as "Activity-Group-Resource" or "Participant-Role-Activity", i.e. represent a script as "a set of activities to be realized by groups sharing some resources" or as "a set of participants playing roles associated to activities". This allows teachers to adopt the structure that best fits their contextual and/or personal needs or perspective.

However, a major limitation of a table is the representation of complex sequencing of activities. Tables allow for representing linear sequences. Representation of conditions, parallelism, or loops and, more generally, intuitive representation of the dynamic dimensions of scripts require languages building on graph models.

Designing script editors leads to a classical dilemma when designing representation structures. Proposing table representations is advantageous because usability experiments and analyses of teachers' basic ICT skills show they are very easily understood and used by non-trained teachers (Sobreira & Tchounikine, 2012). However, they lack some representation power (sequencing aspects). On the other hand, workflow representations are more powerful, but require specific training. And although it does make sense to attempt to improve teachers' skills by offering advanced tools and specific training, providing teachers with such tools may be an obstacle to adoption out of lab experiences or niche settings.

One approach to this issue is to combine the advantages of multiple representations by interoperating tools. It is possible to enhance a table editor by sequencing representation means, but this breaks the simplicity of the design. As suggested by the Model Driven Engineering movement, a complex construction may rather be addressed via different models/tools offering different perspectives and advantages. In the case of macro-scripts, the advantage of such an interoperation would be to offer both (1) a specific editor providing easy, flexible representation of some aspects of the scripts, and (2) another specific tool for representing and implementing complex sequencing. This is an alternative to attempting to merge different representations within a single interface, with risks of overly-complex or poorly-intuitive interfaces.

Within this perspective, we have researched how to transform a table representation (as produced by the ediT2 editor) into a workflow. In line with the objective of contributing to interoperation of tools, we have used the jBPM model (jBPM html). jBPM is a freely available open source workflow engine that provides building blocks to describe the order in which a series of steps need to be executed. Examples of these building blocks are timers, events, composite nodes, start and end nodes, split and join nodes for branching and synchronization, operation nodes that can be associated with code, fault nodes corresponding to issues in the process, and for-each nodes to implement repetition mechanisms. This keeps the ediT2 table editor focused on what it is designed for (easy and flexible representation), representing other aspects (complex scheduling) in another specific and designed-for tool (here, the jBPM tool), which may itself be interoperated with other tools.

Transforming a table representation into a workflow

Transforming a table representation into a workflow is a model mapping issue. It requires linking the editor's representation structure and the workflow's notions, and generating from the editor a structure that can be read by the workflow editor. We will consider here the Activity-Group-Participant-Resource pattern (a script is defined as a set of activities achieved by groups, these groups being formed of participants that have access to, and produce, resources). ediT2 allows other patterns but, thanks to its formal internal representation, it can automatically transform a representation using a pattern into the one corresponding to another pattern.

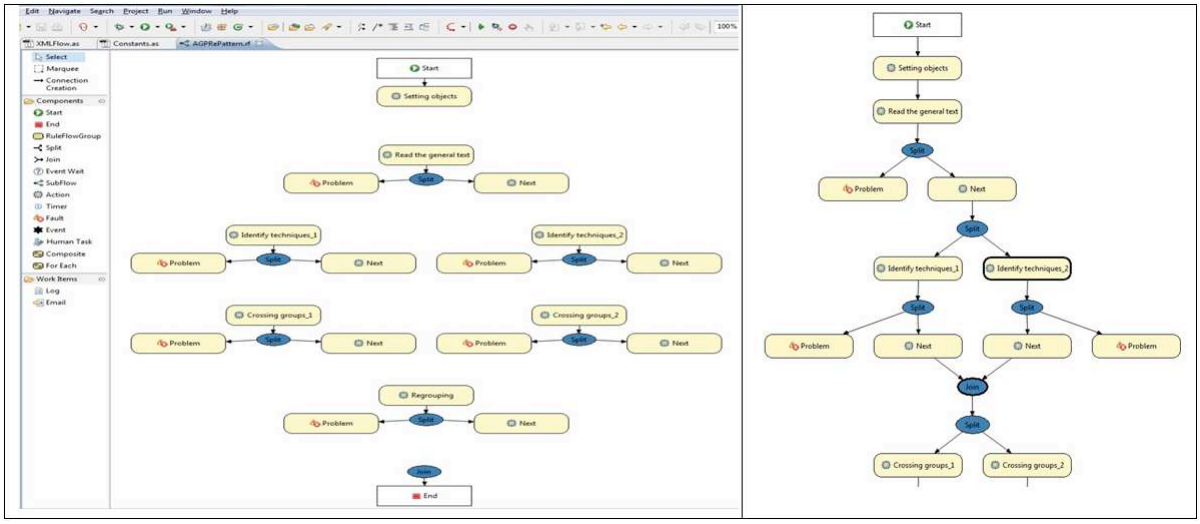

Figure 2. The workflow skeleton generated for the script presented in Figure 1.

The implemented process for transforming a table representation into a flowchart is based on the following algorithm (other solutions are possible). (1) A "Start" node and an "End" node are automatically

created. The "Start" node is linked to a "Setting objects" operation which contains the code automatically generated to set the different objects' values, e.g., the list of students per groups. (2) Each activity is transformed into a set of three operations. The first one corresponds to the activity itself (e.g., "Read the general text"). It is connected via a split connector to a "Problem" operation (which denotes the state to be reached if the activity fails) and a "Next" operation (which denotes the state to be reached if the activity succeeds). (3) When an activity is associated with *m* different groups (or sets of participants), it is automatically broken down into *m* operations which correspond to the *m* instances of the corresponding activity. For example, in the jigsaw script as represented in Figure 1, the "Identify techniques" activity is associated with two groups G1 and G2; this is translated into two instances of the "Identify techniques" operation, one corresponding to G1 and the other to G2. (4) An initial graphical position is set for the different operations and their associated connectors, ordering the operations by the table top-down structure and juxtaposing multiple activity instances if any (see left part of Figure 2). Once the table representation is over, it may be exported by pressing on an "ediT2tojBPM" button. This generates an XML file corresponding to a jBPM representation of the workflow skeleton. Technically, this transformation, which is far from trivial, is implanted by analyzing and manipulating the ediT2 script machine representation, which is a tree data structure. This skeleton can then be visualized and completed (e.g., representing repetition, parallelism or crossing mechanisms) through the jBPM interface manipulation (Figure 2, left side). Figure 2 right side presents a completed version. What is obtained is a language similar to MoCoLADe (Harrer et al., 2009), i.e., a workflow-based language. Going from a workflow representation to a table representation is also possible but would, in general, lead to a loss of information.

Using the workflow representation

A first use of the obtained workflow representation is, basically, to complete the initial design. A second one is to configure an enactment framework. A script represented as a table structure only (i.e., relying on a basic sequencing corresponding to a linear sequence of activities) may straightforwardly be operationalized on repository-like platforms such as Moodle. This may be achieved in different ways, implementing an *ad hoc* mapping or using an intermediate system that can link the editor representation to the framework meta-model, e.g., GLUE!-PS (Prieto et al., 2011). When considering a script based on complex scheduling, however, the operationalization must be conducted from the workflow representation and implemented in a framework that can reify the resources flow (access to data and tools according to the script and not as a group of resources available at any moment). From this perspective, it has been shown how a workflow representation may be used to automatically configure workflow-like platforms such as CeLS (Harrer et al., 2009). Another use of the workflow representation is testing some of the script's characteristics by playing it as a simulation (i.e., indicating the activities flow and their associated features such as users or resources).

Let's consider this simulation aspect. Several authors have raised the fact that teachers could benefit from simulations to refine parameters such as grouping or the flow of activities (Harrer et al., 2009; Weinberger et al., 2008). For instance, given a large number of students, it can be difficult to identify how to pair students. A possible approach is to model students' profiles, form groups, and simulate the script enactment (i.e., make the computer compute the fact that activities "fail" or "succeed" depending on students' profiles). The "fail" and "succeed" criteria may be defined in different ways.

As a proof of concept, we adopted the following trivial modeling (which aims to exemplify the approach and, indeed, is not meant as an advance in students' cognitive modeling). Each student is associated with two normalized values per activity instance: activity-skill and activity-peer-collaboration. In a similar way, each activity instance is associated with two threshold values related to skill and collaboration. Given a group formation (*n* students) and an activity instance, the simulation calculates (1) the group workforce and (2) the group collaboration, and compares them to their respective thresholds to decide if the activity instance is considered as achieved or not:

$$(1)\quad groupWorkforce = \sum_{i=1}^{n} activitySkill_{student_n} \qquad (2)\quad groupCollaboration = \sum_{i=1}^{n} \frac{\sum_{j=1(j\neq i)}^{n} activityPeerCollaboration_{student_i student_j}}{n-1}$$

Given the formulas to be used, the code for the different operations is automatically generated from the ediT2 interface to the jBPM XML representation. Each operation is associated with the corresponding activity's list of students as defined in the ediT2. Therefore, once the different values have been edited the simulation can be launched. The process goes from one node to another following the different join and split connectors, the group-workforce and group-collaboration formulas, the students' activity-skill and activity-peer-collaboration values, and the defined thresholds (all these features being easily modifiable). If an activity fails, the flow reaches the corresponding "Problem" node and stops the simulation; if it succeeds, it reaches the "Next" node. "Problem" and "Next" nodes are associated with some code to print messages on the console. Figure 2 (right side) shows an example of execution. The different operations are highlighted one after the other, whilst messages are printed on the console: "*Activity: Read the general text (Participants: e1, e2, e3 and e4; Resource: General text.IN) performed with success!*"; "*Activity: Identify techniques_1 (Participants: e1 and e2.*

Resources: Insulation text.IN and Insulation list.OUT") performed with success!". In Figure 2 (right side), the process is stopped on the join connector because the activity "Identify Techniques" for G2 has failed; message in the console is: "*Activity: Identify techniques_2 (Participants: e3 and e4; Resources: Heater text.IN and Heater list.OUT) failed because the skill summation (0.7) is less than the workforce threshold (1.0)*". Other groupings (new skill, collaboration, or threshold values) can be edited and tested similarly. The simulation can also be adapted by, for example, modifying the connections and/or the criteria attached to the join or split nodes.

Conclusions

Offering a large variety of tools that can be interoperated is, in our opinion, an important condition for CSCL basic practices to develop. The contribution of this article is to show how teachers may be offered complementary means in the form of a table representation (featuring simplicity and intuitiveness, but arguably lacking means to represent scheduling) and workflow-based representations (featuring data/work flows and scheduling, but arguably more complex). More generally, it goes into the direction of providing teachers a large offer of representations means to choose from. Users interested by an all-in-one framework (but imposing a predefined perspective) may use a LAMS-like system. Users interested by a simple representation may use an ediT2-like system. Users interested by a workflow representation may use a framework natively offering such a representation (e.g., MoCoLADe) or take advantage of a table representation (which offers a different perspective) and then extend it via workflow representation (e.g., using jBPM, but another option could be Freestyler and/or a direct translation into MoCoLADe). Operationalization may be addressed within a repository-like platform (e.g., Moodle) or a workflow-like platform (e.g., CeLS). Figure 3 shows how some of these tools may be interoperated. Effective and seamless interoperation requires engineering work, but research efforts (from which the work presented here) have shown the feasibility.

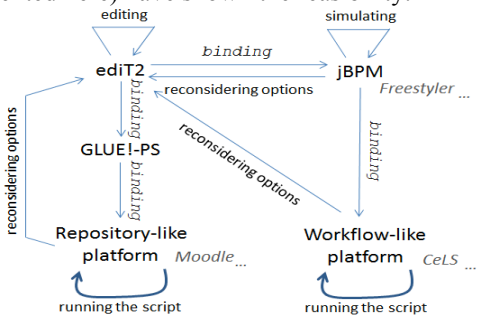

Figure 3. Interoperating different tools.

References

Botturi, L., & Stubbs, T. (2007). Handbook of Visual Languages for Instructional Design: Theory and Practices. *Information Science Reference*, Hershey, IGI Publishing Hershey.

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

Harrer, A., Kohen-Vacs, D., Roth, B., Malzahn, N., Hoppe, U., & Ronen, M. (2009). Design and enactment of collaboration scripts: an integrative approach with graphical notations and learning platforms. *International Society of the Learning Sciences*, CSCL Conference (pp. 198-200).

jBPM 2013. JBoss Business Process Management Suite (www.jboss.org/jbpm), visited in 03/21/13.

Kobbe, L., Weinberger, A., Dillenbourg, P., Harrer, A., Hämäläinen, R., Häkkinen, P., & Fischer, F. (2007). Specifying Computer-Supported Collaboration Scripts. *International Journal of Computer-Supported Collaborative Learning*, 2(2-3), 211-224.

Neumann, S., Klebl, M., Griffiths, D., Hernández-Leo, D., de la Fuente Valentín, L., Hummel, H., Brouns, F., Derntl, M., & Oberhuemer, P. (2010). Report of the Results of an IMS Learning Design Expert Workshop. *International Journal of Emerging Technologies in Learning*, 5(1), 58-72.

Prieto, L.P., Asensio-Pérez, J.I., Dimitriadis, Y.A., Gómez-Sánchez, E. & Muñoz-Cristóbal, J.A. (2011). GLUE!-PS: A Multi-language Architecture and Data Model to Deploy TEL Designs to Multiple Learning Environments. *European Conference on Technology Enhanced Learning* (pp. 285-298).

Reload Editor 2012. Reload: Reusable eLearning Object Authoring & Delivery (www.reload.ac.uk/editor.html), visited in 10/17/2012.

Sobreira, P., & Tchounikine, P. (2012). A Model for Flexibly Editing CSCL Scripts. *International Journal of Computer-Supported Collaborative Learning*, 7(4), 567-592.

Weinberger, A., Kollar, I., Dimitriadis, Y., Mäkitalo-Siegl, K., & Fischer, F. (2008). Computer-supported collaboration scripts: Theory and practice of scripting CSCL. In N. Balacheff, S. Ludvigsen, T. de Jong, A. Lazonder, S. Barnes & L. Montandon (Eds.), *Technology-Enhanced Learning. Principles and Products* (pp. 155-174). Berlin: Springer.