Sobreira, P. & Tchounikine, P. (2012) A model for flexibly editing CSCL scripts. International Journal of Computer Supported Collaborative Learning 7 (4).

A Model for Flexibly Editing CSCL Scripts

Péricles Sobreira[1,2], Pierre Tchounikine[1]

[1]LIG – Université Joseph Fourier, Grenoble-France, Pericles.Sobreira@imag.fr, Pierre.Tchounikine@imag.fr
[2]DCET – Universidade Estadual de Santa Cruz, Ilhéus-Brazil

Abstract:

This article presents a model whose primary concern and design rationale is to offer users (teachers) with basic ICT skills an intuitive, easy, and flexible way of editing scripts. The proposal is based on relating an end-user representation as a table and a machine model as a tree. The table-tree model introduces structural expressiveness and semantics that are limited but straightforward and intuitive. This approach is less expressive and introduces less semantics than approaches based on workflow representations and complex meta-models. However, it may be enhanced to represent complex features such as by-intention grouping mechanisms, constraint checking or configuration of enactment frameworks. A usability test suggests that the model/interface is easy to use and that teachers avail themselves of the flexibility available to model scripts according to their perspectives.

Keywords:

CSCL Scripts, Editing, Flexibility.

1. Introduction

A Computer-Supported Collaborative Learning (CSCL) script is a learning scenario introducing structure and constraints that guide how a group of distant or co-present students collaborate. CSCL scripts raise a variety of research issues: elaborating efficient scripts and design-principles (Aronson *et al*. 1978, Palincsar & Brown 1984, Dillenbourg & Hong 2008), studying the issues arising from their use by practitioners (Hernández-Leo *et al.* 2006), studying advantages and risks such as limiting the occurrence of certain kinds of negative processes (O'Donnell 1999), improving learning outcomes (Weinberger *et al.* 2010), constraining collaboration in a way that may inhibit natural peer interaction (Dillenbourg 2002) or conflicting with students' internal scripts (Weinberger *et al*. 2008). One of these research issues is the operationalization of scripts.

Operationalization of scripts is "the process of going from an abstract and technologically-independent description of the script to the effective setting the students will be presented with, i.e., the precise description of the tasks, groups, constraints to be respected, and technological setting to be used" (Tchounikine 2008). Since the *ijCSCL* announced "Scripting in CSCL" as a "flash theme" (Stahl & Hesse 2007), various topics related to operationalization have been addressed in the journal. Such topics include identification of conceptual primitives to model and represent scripts (Kobbe *et al*. 2007), review of issues and a general model for operationalization of scripts in technical settings (Tchounikine 2008), analysis of the mechanics of scripts (Dillenbourg & Hong 2008) and an approach for implementing a given script in a broad variety of platforms (Wecker *et al.* 2010).

In this article, we continue this on-going effort to study operationalization issues, focusing on a specific concern: offering representation means that (1) are sufficiently simple and flexible to allow teachers to edit a script and adapt it to their view and context, and (2) can be extended to match other needs or implement advanced features such as representing complex grouping mechanisms, checking constraints or configuring the software students will be offered. In this work, *editing a script* is to be understood as adapting the script phases (e.g., breaking an activity into several activities or changing a role), instantiating the script with the relevant resources and/or setting up parameters such as group composition. Target users are teachers with basic ICT skills but no particular training in CSCL methodology.

The contribution we propose is an approach based on relating (1) an easy-to-use end-user representation in the form of a table with (2) a machine model of the script as a tree. The table-tree model introduces structural expressiveness and semantics that are limited but straightforward and intuitive. The table presentation enables direct manipulations such as

moving a student from one group to another or splitting an activity (script flexibility). It also makes it possible to change how the notions provided to represent a script are used, i.e., changing the representation pattern (representation flexibility). This approach is not meant to replace the mainstream approach consisting in offering a graph-based representation of scripts based on a meta-model detailing the different conceptual aspects of scripts. Rather, it offers an alternative featuring simplicity, easiness and flexibility, to be used if and when needed. This basic model offers limited expressiveness and is permissive, i.e., does not control users' representations in the same way as a meta-model approach. However, if this appears to be an issue, it may be overcome by enhancing the basic model with extensions.

This article is organized as follows: Section 2 presents the background of this work, i.e., recalls CSCL script basics and the considerations we build on. Section 3 presents and illustrates the proposed model, and shows how it may be enhanced to represent complex mechanisms (e.g., complex grouping mechanisms), check specific constraints or configure the enactment platform available to students. Section 4 presents the results of a usability test. Section 5 discusses the core characteristics of this approach (relation to the overall operationalization issue, expressiveness and limitations, permissiveness, rationale for the tree-table representation and interface variants). Finally, Section 6 draws conclusions and presents perspectives.

2. Background

Research issues related to script operationalization include identifying requested specifications, designing and creating adequate technology, verifying that this technology matches its specifications, analyzing how it is used in lab and in common practices, and analyzing the outcomes. The work presented in this article is a software-centered contribution related to one particular phase of this long-term effort, namely designing and creating adequate technology for a specific goal (editing scripts). In this section, we briefly recall the rationale and bases for this work, and the perspective it builds on. These elements will be taken as standpoints and will not be discussed as such (the point is not to present new theoretical considerations related to why scripts and script flexibility are of interest, but to study how to design and implement technology in line with this goal). Additional discussions are presented in Section 5.

A CSCL script is a scenario that aims at setting up conditions (guidance and constraints) that may improve the likelihood that knowledge-generating interactions such as explanations and engaging in argumentation, negotiation, conflict resolution, or mutual regulation occur (Dillenbourg & Jermann 2007, Kollar *et al.* 2006). To do so, scripts structure the setting by defining precise sequences of activities, grouping students by specific criteria, creating roles or constraining the mode of interaction among peers (Fischer *et al*. 2007, Kobbe *et al*. 2007). Typically, a group of students is given a task. This overall task is then broken down into subtasks, where the outputs of some subtasks are a resource requested for others: students are given different roles and resources, and these subtasks, dataflow and division-of-labor are studied to create a context within which students interact. Scripts may be defined and implemented as in-presence scripts (involving students present in the same classroom), on-line (involving distant students addressing the proposed tasks and communicating via a computer-based system only), or blended. As quoted in (Wecker *et al*. 2010), a broad range of studies highlighting how scripts fostered learning activities has been published (e.g., Baker & Lund 1997; De Wever *et al*. 2009; Kollar *et al*. 2007; Rummel & Spada 2005; Schellens *et al*. 2007; Schoonenboom 2008; Slof *et al*. 2010; Stegmann *et al*. 2007; Weinberger *et al*. 2005; Weinberger *et al*. 2010).

Concerning students, operationalizing a script requires offering students a *technological enactment framework* that will help them to coordinate and successfully perform the script's tasks. Such a framework typically includes features such as communication tools, awareness tools or means to share resources and, when needed, specific pieces of software related to the task at hand. Enactment frameworks may correspond to different realities. Examples are a piece of software specifically dedicated to a specific script or script-family and implementing part of the script-specific support and constraints, e.g., the specific systems implementing the Argue-Graph or ConceptGrid scripts (Dillenbourg & Hong 2008), a complex generic environment whose features can be organized to fit a given script's needs, e.g., the CeLS environment (Ronen *et al.* 2006), a platform providing support on top of diverse Web contents (Wecker *et al.* 2010), or a basic Learning Management System (LMS) offering general features, e.g., Moodle (Moodle 2012).

Concerning designers and teachers, there is no consensus on a precise life cycle but, when analyzing different proposals (e.g., Dillenbourg & Tchounikine 2007, Tchounikine 2008, Weinberger *et al*. 2008, Villasclaras-Fernández *et al*. 2009), the following dimensions can be identified. *Script design* is the identification of the script's principles, i.e., how the learning activities lead to intended learning goals, and important conditions to be respected. Usually, design is addressed by instructional designers, and scripts described in a general and more or less abstract way featuring their core mechanisms: see for example scripts introduced in (Kobbe *et al*., 2007) or patterns in (Hernández-Leo *et al*. 2010). *Script editing* is the pre-session process required to create a *script instance* adapted to the setting (in some works, script editing is referred to as "instantiation"). Script editing is contextual and, usually, addressed by teachers. During the session, while the script unfolds, teachers are also involved in *script monitoring* and *run-time management*.

Script editing content is largely dependent on script granularity level. Although a continuum exists, scripts are often dissociated into micro-scripts and macro-scripts (Dillenbourg & Tchounikine 2007). Micro-scripts are studied at a psychological level and aim at scaffolding students' process at the interaction level. Examples are to ask a student to state a hypothesis and prompt a peer to produce counter-evidence, constrain interactions by prompting turn taking or imposing an argumentation grammar. The core mechanism is often reified within the enactment framework e.g.,

typically, requiring students to use a structured chat or argumentation graphical tool. Such scripts leave little room for options, and editing is merely limited to the setting of learning domain resources and parameters such as group composition. Macro-scripts are pedagogy-oriented large-grained scripts, based on indirect constraints generated by the definition of the sequence of activities or the group characteristics. For instance, the jigsaw macro-script pattern is as follows: first, participants individually work on a topic; second, students having worked on the same topic meet in "expert groups" to exchange ideas; finally, "jigsaw groups" are formed by grouping students who each worked on a different topic in the preceding phase (Hernández-Leo *et al.* 2010). This general principle, which may be used in very different settings (e.g., for a two-hour or a two-month session, with four or forty students), leaves room for many options. At least the script must be instantiated with the learning domain (e.g., if the domain is "energy saving", deciding to create groups focusing on "insulation" and others focusing on "heating", and defining the adequate resources as on-line documents, wikis or quizzes), and group compositions must be set up. However, for such macro-scripts, script editing may also involve adapting or increasing script accuracy by taking into account the contextual aspects revealed by local analysis of issues such as related past or future classroom activities, availability of resources, constraints related to the given enactment framework students are familiar with, or which is imposed by the setting, or other dimensions such as time pressure or institutional demands (e.g., the fact that each student must receive an individual grade). Management of macro-scripts is now often conceptualized as part of the general *orchestration* of the teaching setting. Within this perspective, a script is to be seen as only one element of the setting, and the setting is to be seen as a complex, multilayered and highly constrained ecosystem whose "orchestration" may require some run-time adaptations (Dillenbourg *et al*. 2011).

Design/editing and micro/macro dichotomies open up perspectives for different views with respect to what may be considered as *legitimate variations of scripts*. One perspective is to consider that a script defined by instructional designers and positively evaluated in research should not be adapted by teachers any more than is needed to instantiate it with the learning domain and to deal with situations in which it cannot be straightforwardly implemented (e.g., if class size is not a multiple of intended group size). Another perspective, that we will refer to as the *open perspective to script editing*, is that teachers, as those in charge of reflecting on and managing the classroom (the students, the overall activities, the institutional context, etc.) and script enactment, may engage in substantial adaptations, before and during the session (orchestration perspective). With respect to micro-scripts, the former perspective is required. With respect to macro-scripts, the question is more open, if only because these scripts are more general constructions and thus require more instantiation work. The open perspective covers a continuum ranging from limiting adaptations within the script's intrinsic constraints (i.e., its design rationale principles (Dillenbourg & Tchounikine 2007)) to unrestricted adaptations.

Another aspect of the fact that teachers may be allowed to engage in extended edition of scripts is teachers' appropriation. Adaptability to teachers' settings and perspectives are important characteristics for appropriation and usage (Williams *et al.* 2004). This aspect, however, is dependent on many other dimensions, including teachers' skills and institutional dimensions: in some settings/countries, instructional designers define scripts while teachers, at the most, instantiate and set up the script. In other places, teachers may have a more active role, including the design of variants or even new scripts. More generally, the fact that a learning design language makes it difficult for teachers to easily adapt a scenario has been identified as a possible issue for adoption (Neumann *et al.* 2010).

In this work, we consider macro-scripts and the open perspective to script editing. As suggested in (Dillenbourg & Tchounikine, 2007), we consider script editing as a teacher's task that consists in taking local decisions: this ranges from the general specification to the precise instantiation of specific aspects related to the field under consideration, specific goals, individual student skills or profiles, time constraints, space, available technology, pre- and post-activities, or institutional dimensions. Moreover, we give specific importance to the fact that teachers are empowered in viewing and representing the script according to their perspective.

Script editing, as considered here, is an elaboration experience rather than a straightforward process. While tuning the script support/constraint aspects, different registers may be used: the precise task students will be given, how this task is broken down into subtasks, the way the outputs of some subtasks become a resource for others, the way students are grouped (taking into account general principles and students' effective characteristics), how roles are emphasized or faded, etc. Although the structure and some decisions are imposed by the script rationale (e.g., creating jigsaw groups) and other decisions may be imposed by the setting, a variety of options may be envisaged and sketched while the editing process, considering the different options and refining decisions, until a satisfactory structure has been obtained.

Script editing is a specific concern related to the more general topic of script representation (Miao *et al.* 2005, Botturi *et al*. 2006, Harrer & Hoppe 2008). Considering representation tools and editing in particular, two general features of interest may be highlighted. A first feature is to offer users relevant means to reflect on the script. Different users may have different needs and requirements. While instructional designers may require means to specify scripts or index them in repositories, teachers are more interested in how to adapt and efficiently deliver scripts in real classrooms (Weinberger *et al*. 2008). A second feature is to configure (or partially configure) the enactment framework, i.e., from the representation of the script as defined within the teacher-oriented editor, configure the constructions denoting the groups and resources (etc.) in a target platform such as Moodle (Moodle 2012), LAMS (LAMS 2012), CeLS (Ronen *et al.* 2006) or CoFFEE (Belgiorno *et al.* 2008). If the script editor is interoperated with the enactment framework and can retrieve data related to script unfolding, this feature may also be used to allow teachers to adapt the script and/or enactment framework at run-time, in relation with overall orchestration of the setting.

3. The T^2 model

3.1. Overview

The T^2 model is an intentionally simple model designed to facilitate editing and adaptation of macro-scripts within an open perspective to script editing. Its main goal is to (1) provide a base for creating editors offering intuitive and easy-to-use means to edit the script while (2) keeping open the possibility to enhance the editor with advanced means or mechanisms if/when considered useful. The prototypical use-case considered here is that of a teacher with basic ICT skills and no particular CSCL methodology training, taking into account an already designed script such as those presented in Figure 2 (Section 4) and, while editing it (i.e., defining the precise activities, resources or groups), adapting it to his/ her view and context.

The research goal is to explore a perspective significantly different from the mainstream approach. With respect to macro-script representation, the mainstream approach is to define a conceptual meta-model of a script (e.g., the meta-model presented in (Miao *et al*., 2005)) and to develop a language/editor implementing this meta-model expressiveness and semantics, i.e., requiring that users comply with the model constraints. Most of these languages/editors adopt a graph-based approach (Botturi & Todd 2007), which neatly captures scripts dynamics (dataflow and/or workflow) and allows a straightforward link with workflow engines (Harrer *et al.* 2007). Not misunderstanding the interest of this approach, in terms of expressiveness and conceptual support in particular, our work explores an alternative building on different premises. First, we take as an entry point of the work and main design decision offering a language/editor based on a table, as a basic and very common structuring device. Second, we consider basing the model semantics on the table structure (i.e., the column/row structure) rather than on a proper meta-model (we will refer to this as a *structural semantic*). We manage these specifications by placing the interface (the table) in relation with a machine-readable model as a tree. This principle makes it possible to address editing of the script via natively-simple manipulations (mouse-manipulations of a table structure and content, as in office suites), instead of elaborating a rich and complex language hypothetically rendered easy to use by basic users via user-friendly interfaces and training.

This structural approach is highly flexible, of a somewhat limited expressiveness, and permissive. The basic model does not directly allow representation of complex mechanisms and implements an unrestricted perspective to script editing. However, it can be enhanced by introducing more expressiveness and support/constraints on top of the structural features. Introducing such extensions is to be thought of as a way to benefit from the model native simplicity and flexibility advantages while enhancing it or overcoming some of its specificities if/when they appear as limitations (using extensions to obtain an equivalent to current state-of-the-art language building on graph representations and a powerful meta-model is of little interest as a meta-model approach is much more straightforward).

The model presentation is structured as follows: introduction of the Table/Tree structure (§3.2), presentation of the basic semantics (§3.3), the flexibility offered (§3.4), the model constraints (§3.5), how machine representation is managed (§3.6), then, finally, presentation of how the model may be enhanced to extend expressiveness (§3.7) and semantics (§3.8), and how it allows an enactment framework to be configured (§3.9). Illustrations are taken from the usability test (presented in Section 4). Further discussions are proposed in Section 5.

3.2. The Table / Tree structure

The T^2 model proposes considering a CSCL script as (1) a *table* composed of *columns* and *rows* (which can be broken down into *sub-rows*) if it is addressed via its user-oriented visualization, and (2) an *n-ary tree* composed of *levels and branches* if it is addressed as a formal structure. The model is named T^2 to denote this double Table and Tree structure. From a Computer Science perspective, the table and the tree respect a structural correspondence, i.e., one element of the tree corresponds to one element of the table interface and *vice versa*.

Tables are two-entry structures. We have defined the column as the entry denoting conceptual notions: each column corresponds to a notion (rows thus have a homogeneous column structure). In our current work, we reuse the notions proposed in (Kobbe *et al*. 2007): *activities* (what must be done), *participants* (individuals), *groups*, *roles* and *resources* (virtual or physical objects used, modified or produced by the participants). The rationale for using this framework is that it has been elaborated as a consensus by scientists from several fields (educational, cognitive and computer sciences) as a result of their experiences in specifying CSCL scripts, has proven it can model a variety of macro-scripts of different types, and is fairly intuitive (as confirmed by our usability test). If we take the example of an interoperation with an enactment framework such as Moodle, resources may be an instruction page, a file, a quiz, a chat/forum, a wiki or an URL to a Web-resource. With respect to the original framework, we add the option of tagging resource labels to identify whether they are offered (-in) or produced (-out) by students.

Figure 1 presents (1) the general interface of a standalone editor (named ediT2) that we have developed to analyze whether teachers were at ease with the model, and (2) the corresponding machine representation as a tree. In the editor, the zone to the left is an *ad hoc* feature used to create the items (activities, groups, participants, resources and roles) that will be referred to in the script[1]. Let us consider the two first rows. The teacher has first represented a "Read the general text" activity, to be carried out by each of the four students (e1, e2, e3 and e4), where "General text" is the available

[1] If such an editor is used when integrated with a LMS, some of these items may be downloaded from the LMS database.

resource. The second activity, “Identify techniques”, involves two groups (G1 and G2) with different participants (e1 and e2, e3 and e4, respectively) and different resources. In the tree representation, a fictional root is defined. Its children are the elements corresponding to the first column in the table, i.e., the four activities “Read the general text”, “Identify techniques”, “Crossing groups” and “Regrouping”. “Read the general text” has a linear structure (branching factor = 1). “Identify techniques” has two children, G1 and G2 (branching factor = 2). In this case, none of the cells corresponding to the participants and resources are broken down any further, and the role notion is not used.

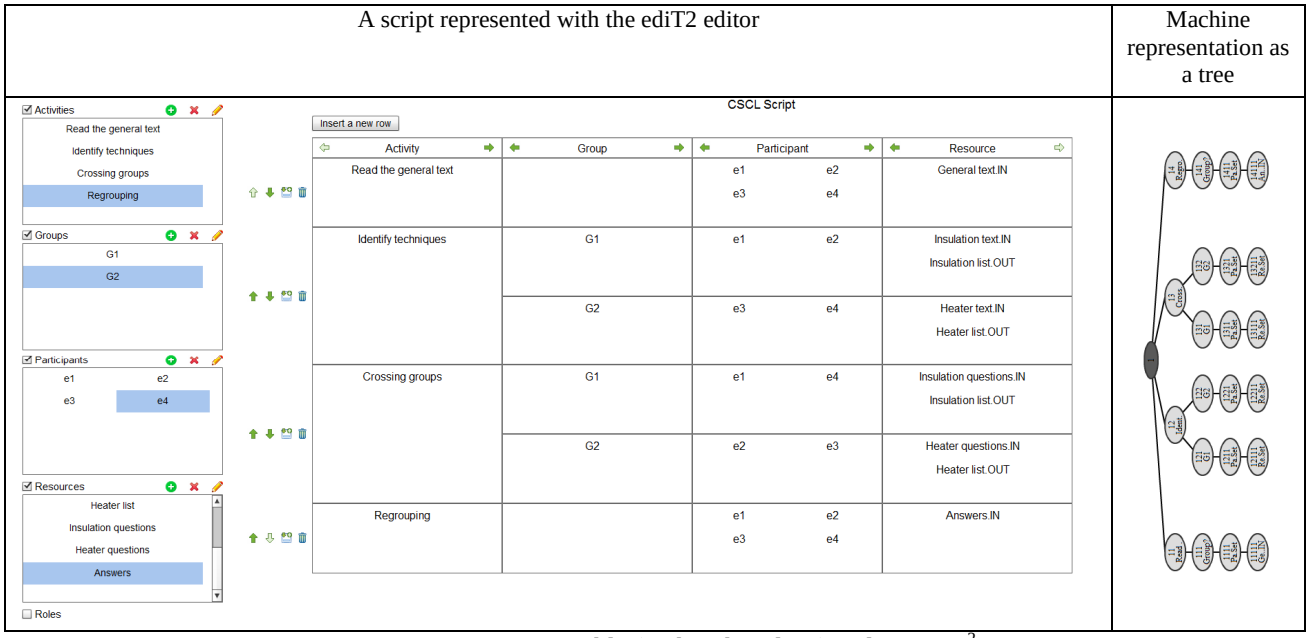

Figure 1. A script as a table (within the ediT2) and as a tree[2]

We will call *script-structure* the ordered list of notions used as columns (e.g., in Figure 1, Activity-Group-Participant-Resource), *pivotal notion* the notion used as the first column (Activity), *script-component* a row, referring to it by its pivotal notion (e.g., the “Read the general text” script-component), and *items* the specific values attached to the cells or, within the tree perspective, to the nodes (e.g., participant “e1”).

Representing and editing a script is a double-dimension process. One dimension is related to the definition of the script-structure, i.e., the model that rules the script-components description. At this level, the pivotal notion defines a modeling commitment (e.g., in Figure 1, a script is modeled as a set of Activities). The second dimension is related to the script-components, i.e., creating and manipulating rows and items. Table 1 summarizes basic actions to represent/edit a script.

Table 1. Basic actions to represent/edit a script

	Example	Table perspective	Tree perspective
Representing/editing the script-structure	Deciding that a script will have an Activity-Group-Participant-Role-Resource structure	Creating and ordering the columns of the table	Defining the structure of a branch
Representing/editing the script-components	Creating the “Read the general text” script-component	Creating a row	Creating a branch
	Associating “e1” with group “G1”	Associating items with a cell	Associating items with a node
	Associating “G1” and “G2” with Activity “Identify techniques”	Creating sub-rows (or merging cells of existing rows) and then associating items with the cells	Creating a sub-branch or unifying sub-branches and associating items with the nodes

3.3. Basic semantics

Natively, the column headings are just type labels. An *activity* column states that the values that can be placed in this column are edited as the type *activity* (i.e., in the ediT2 interface, are defined as activities in the activity box which is to the left of the interface) A script is a list of script-components, where a script-component is a homogeneous

[2] In this Figure, we have re-written the representation created by one of the teachers who participated in system tests for the jigsaw script presented in Figure 2, and translated the items (names of activities, etc.) to English. One may notice that for the “Regrouping” activity (last row) the teacher tagged the “Answers” resource as “.in” whereas it is likely to be, rather, an “.out” resource.

juxtaposition of instances of notions (in our case: activities, participants, etc.). The notion' semantics is only carried out by the label and its intuitive meaning (and, thus, possibly related to the user's perspective, which may be idiosyncratic and/or related to professional training or context of practice). How an activity relates to groups or roles is only natively represented by a generic *is-associated-with* relationship denoted by the row structure (structural semantics).

Table 2 presents a basic interpretation of the generic *is-associated-with* relationship. As an example, within this interpretation, the "Identify techniques" script-component represented in Table 1 may be read as follows: the activity "Identify techniques" *is achieved by groups* G1 and G2; G1 *is composed of* e1 and e2; these participants *are presented with* the resource "Insulation text" and *are to produce the resource* "Insulation list"; G2 *is composed of* e3 and e4; these participants *are presented with* the resource "Heather text" and *are to produce the resource* "Heather list".

Table 2. A basic interpretation of the *is-associated-with* generic relationship

	Activity	**Group**	**Participant**	**Role**	**Resource**
Activity		The activity is achieved by group(s) …	The activity is achieved by participant(s) …	The activity is achieved by playing the role(s) …	The input/output resource(s) of the activity is (are) …
Group	The group is to consider the activity(ies) …		The group is composed of the participant(s) …	The group is to play the role(s) …	The group is presented with / is to produce the resource(s) ...
Participant	The participant(s) is (are) to consider the activity(ies) …	The participant(s) will pair as group(s) …		The participant(s) is (are) to play the role (s) …	The participant(s) is (are) presented with / is (are) to produce the resource(s) …
Role	The role involves considering the activity(ies) …	The role will be played by the group(s) …	The role will be played by the participant(s) …		The role is to be played using the input resource(s) / producing the output resource(s) …
Resource	The activity(ies) to be considered in relation to the resource(s) is (are) …	The group(s) working with this (these) resource(s) is (are) …	The participant(s) working with this (these) resource(s) is (are) …	The role(s) to be played with this (these) resource(s) is (are)…	

This structural approach is significantly different from explicitly representing relationships such as *is-achieved-by* or *is-performed-by*, and associated constraints, using a proper meta-model defining a script (see for instance (Miao *et al.* 2005)). From a meta-modeling perspective, the implicit T^2 meta-model is basic: a list of notions N_i, where N_i is related to N_{i+1} by a 1..* relationship and, in our case, $i \in [1.5]$, and N_i assumes values in {"activity, "group", "participant", "role", "resource"}. Such a perspective, however, does not capture much of the work's rationale. Another way to phrase it, more in line with the approach, is to state that a table is an easy-to-use way of offering end-users (limited) meta-modeling means, in this case selecting the notions they want to use and ordering them as they prefer.

3.4. Flexibility offered

The original flexibility feature introduced by the approach, which we refer to as *representation flexibility*, consists of the possibility of determining the script-structure, i.e., which notions are used and in what order. This allows for addressing scripts conceptualized in very different ways. A prototypical script-structure is Activity-Group-Participant-Resource: a script is addressed as a set of activities to be carried out by a group (or groups) composed of participants, where each participant is associated with resources. This pattern has many variants: specifying roles for participants and associating roles with specific resources (e.g., Activity-Group-Participant-Role-Resource), associating resources with activities (e.g., Activity-Resource-Group-Participant), etc. Other very different examples are: Group-Role (a script is addressed as a set of groups and the roles to be played by these groups) or Resource-Group-Activity (a script is addressed as a set of resources which will be used by different groups to conduct different activities).

For a given script modeled within T^2, different types of editing actions can thus be discerned:

1. Editing a script-component (e.g., adding a participant to a group). In terms of the model, such changes only affect the items attached to a cell/node. They do not affect the tree structure.
2. Editing the script-component list (e.g., adding/removing a script-component or swapping script-components). In terms of the model, such changes affect the number of tree branches or their ordering.
3. Editing the internal structure of a script-component (e.g., defining new roles within an activity by splitting a cell). In terms of the model, such changes affect the number of branches representing a script-component (the arity of this branch as a sub-tree), and their associated items.
4. Editing the script-structure (representation flexibility, e.g., using a new notion, removing an existing notion or changing the order of the notions). In terms of the model, such changes affect the structure of the branches.

Referring to the ediT2 interface, the different editing actions correspond to the manipulation possibilities conventionally offered by a table:

1. *Dragging and dropping* an item from one cell to another (e.g., moving a participant from one group to another).
2. *Adding*, *removing* or *displacing* a row (e.g., where Activity is the pivotal notion, adding a new activity or swapping two activities). A Script-component is created by clicking on the "Insert a new row" button (Figure 1) and dragging and dropping items into its cells. Each row is associated with arrows to move them up or down and a bin icon. Several facilities (e.g., duplicate a complex row, with or without its items) are proposed.
3. *Splitting a cell into several cells* (e.g., splitting an activity into two activities or associating an activity with several groups) or *merging cells* (e.g., regrouping individual participants initially spread into different groups). Merges and split features are available when right-clicking on the relevant cell(s).
4. *Adding*, *removing* or *displacing* a column (e.g., changing from an Activity-Group-Participant-Resource perspective to a Role-Participant-Resource perspective). Deciding to use a given notion (i.e., creating a column) requires ticking the relevant box in the left part of the interface. This creates the relevant column in the table (e.g., the "Activity" column). Each column is associated with a left and right arrow moving the column (the column is moved as a whole, i.e., for all already existing rows, which is the application of the script-component homogeneity principle).

It should be pointed out that the model allows changing the script-structure while the script is being edited, i.e., while the table is already filled. If the script-components all respect a 1-1 relationship (i.e., a table made up of full-rows, with no split cells), the order of notions denotes the way the script is conceptually addressed (e.g., as a set of activities or as a set of roles), but the fact that columns are displaced does not further change the script semantics as the 1-1 relationships remain identical. In direct contrast, if the script presents 1-n with $n > 1$ relationships, a change of perspective resulting in column displacement may deeply affect the tree structure (see Section 3.6).

Within our open perspective to script editing, we consider representation flexibility as an important feature allowing teachers to adopt the perspective they prefer. Options to the basic model are to offer even more flexibility and allow end-users to define their own notions or, on the contrary, limit manipulation possibilities (see Section 3.8).

3.5. Constraints

To avoid ambiguous constructions, the constraint introduced by the T^2 model is that a script-component must comply with a 1-n with a $n \geq 1$ relationship (or, in other words, a script is a tree and not a graph: a cell has only one parent). Table 3 illustrates the rationale for this constraint. Given the fact that participants P1 and P2 will engage in activity A1 with the same resource R1 (representation of the left side of the Table), the resource cells could be merged (representation of the right side). However, this would lead to an ambiguous construction with respect to roles (e.g., it is unclear whether P1 or P2 are to play roles RL1, RL2 and RL3). This constraint keeps the table equivalent to a tree and thus interpretable within the adopted structural semantic[3].

Table 3. Rationale for the 1-n with $n \geq 1$ relationship

interpretable construction

Activity	Participant	Resource	Role
A1	P1	Rl	RL1
	P2	Rl	RL2
			RL3

ambiguous construction (it is no longer a tree)

Activity	Participant	Resource	Role
A1	P1	Rl	RL1
			RL2
	P2		RL3

It should be noticed that this constraint applies to the structure of the script-components. It does not mean that items should be referred to in the script only once. Let us consider the case of a script using two resources, a quiz Q1 and a wiki W1 (taking Moodle as the deployment platform). Q1 and W1 may be referred to in different rows. For instance, a first row may state that a first pair of participants P1 and P2 will be offered Q1 and W1, and a second row state that a second pair P3 and P4 will be offered the same resources Q1 and W1. In such a case, all four students will be offered Q1 (individual quiz), P1 and P2 will be able to collaborate via a first instance of wiki W1, and P3 and P4 will be able to collaborate via a different second instance of W1 (in other words: P1 and P2 will not access to P3 and P4 interactions, and *vice versa*). If the four participants' cells are merged, then the four of them will be involved in the same wiki.

From the point of view of the model, the fact that cells are not associated with any value is not an issue (independently from the fact that it is meaningful or not). For instance, an individual phase may be represented by creating a row mentioning the activity, the participant and the resource, where the group column is kept blank. During our usability test, several teachers straightforwardly defined such representations, although we had not made this

[3] Of course, interface subtleties could be imagined to locally overcome this constraint, but this is of little interest and would contradict the simplicity principle. See further discussions in Section 5.4.

explicit during the editor presentation. An example of meaningless construction would be a collective activity not indicating who the participants are.

3.6. Management of the model and users' actions

Since the approach does not offer the semantic advantages provided by a specific meta-model, it may be considered that the T^2 model and its flexibility features can be obtained by using a basic table editor as in office suites. The interest of the model and a specific implementation, however, is to offer services related to tree representation (editing facilities, unambiguous constructions, and possibility of enhancing basic model expressiveness and semantics).

From an end-user perspective, the advantage of the table structure is to avoid complex syntactical constructions, the constraints of which must be understood and properly used: the table structurally denotes the semantics. From a machine perspective, the advantage is similar. Natively, script manipulations correspond to manipulations of branches and nodes. They are interpreted with respect to the tree structure, and reported in the tree representation (or inhibited if inconsistent with the model). This corresponds to basic algorithms on tree data-structures, where these algorithms are uniform (splitting an activity or a group both correspond to the same internal manipulations).

A certain number of algorithms are trivial. Managing (attaching, removing or moving) values associated with nodes is basic. Adding, removing or displacing rows corresponds to manipulations of the list of branches. Adding or removing a column corresponds to the creation or destruction of the corresponding segment in all the tree's branches.

Splitting/merging cells (creation/merges of sub-branches) and displacing a column both require propagating the modifications via tree manipulation algorithms to maintain the representation coherent with the model. Table 4 presents two cases of propagation actions. It may be noticed that, when propagating changes in the tree structure, the sub-rows and items may be managed in different ways. For instance, when splitting the {P1, P2, P3, P4} Participant cell into two (Table 4, row 1), the resulting Participant cells may be reinitialized to blank, associated to the participant(s) preexisting to the manipulation or spread over the two new cells. This may be configured and, anyway, the result may be easily adapted by further merge/split or drag-and-drop actions.

Table 4. Examples of manipulations of script representations and propagations

<table>
<tr><th>Script representation</th><th>Manipulation</th><th>Script representation modified (when applied)</th></tr>
<tr>
<td>
<table>
<tr><th>Activity</th><th>Participant</th><th>Role</th><th>Resource</th></tr>
<tr><td rowspan="6">A1</td><td>P1</td><td>RL1</td><td>R1</td></tr>
<tr><td rowspan="5">P1 P2 P3 P4</td><td rowspan="5">RL1
RL2</td><td>R2</td></tr>
<tr><td>R3</td></tr>
<tr><td>R4</td></tr>
<tr><td>R5</td></tr>
</table>
</td>
<td>Split the cell containing {P1,P2,P3,P4} Participants in two</td>
<td>
<table>
<tr><th>Activity</th><th>Participant</th><th>Role</th><th>Resource</th></tr>
<tr><td rowspan="6">A1</td><td>P1</td><td>RL1</td><td>R1</td></tr>
<tr><td rowspan="3">P1 P2 P3 P4</td><td rowspan="3">RL1
RL2</td><td>R2</td></tr>
<tr><td>R3</td></tr>
<tr><td></td><td></td><td>R4</td></tr>
<tr><td></td><td></td><td>R5</td></tr>
</table>
</td>
</tr>
<tr>
<td>
<table>
<tr><th>Activity</th><th>Participant</th><th>Role</th></tr>
<tr><td rowspan="3">A1</td><td>P1</td><td>RL1</td></tr>
<tr><td rowspan="2">P2</td><td>RL2</td></tr>
<tr><td>RL3</td></tr>
</table>
</td>
<td>Displacement of the Role notion towards the left (or of the Participant notion towards the right)</td>
<td>
<table>
<tr><th>Activity</th><th>Role</th><th>Participant</th></tr>
<tr><td rowspan="3">A1</td><td>RL1</td><td>P1</td></tr>
<tr><td>RL2</td><td>P2</td></tr>
<tr><td>RL3</td><td>P2</td></tr>
</table>
</td>
</tr>
</table>

3.7. Enhancing expressiveness

We shall now examine how the model basic expressiveness may be enhanced. For this purpose, we will refer to Kobbe and his colleagues' framework, which highlights three important mechanisms. *Group formation* specifies how groups of participants are constructed. *Task distribution* specifies how components (e.g., activities, roles or resources) must be distributed among participants or groups. Finally, *sequencing* specifies how the script's phases or tasks will be distributed over time. This can correspond to a simple linear ordering (phases or tasks are to be taken one after the other as listed in the script) or to complex dynamic structure such as traversion, rotation or fading.

Group formation and task distribution

Within the basic T^2 model, group formation and task distribution are represented by extension, i.e., listing the items associated with nodes (see preceding examples). While this may be sufficient in many cases, it may be an issue in others. First, if the script involves many students, creating these lists may become time consuming and/or over complex. For instance, creating groups for a jigsaw script requires mixing students from different focus groups, and scripts such as the reciprocal teaching script (Palincsar & Brown 1984) require the rotation of roles among students during unfolding. Managing these mechanisms for 4 or 6 students is easy, but may become intractable for 20 or 200 students. Second, listing the items associated with a node does not allow specifications as an abstract principle (e.g., "G3 is made up by mixing G1 and G2 students") or a dynamic principle (e.g., "G2 is made up of the five students who finished

activity A1 first"). Such a dynamic and possibly run-time management of groups may be processed by the teacher if he/she controls the enactment scheduling, dragging-and-dropping a student from one group to another "by hand". However, when considering large groups or automated run-time management, representation by extension is an issue.

Overcoming the "by-intension description" limitation does not need to modify the bases of the T^2 model but, rather, its implementation. If considering the ediT2 implementation, it must be enhanced by associating nodes with algebraic constructions and introducing configuration interfaces in the groups (etc.) definition boxes. For instance, the Universanté script (Dillenbourg & Jerman 2007) requires, in some places, to refer to groups studying similar clinical cases and, in others, to groups made up of students from the same/different countries. Representing such scripts may be addressed by representing the necessary information in the participant data-structure (e.g., country) and modify the group definition box to allow defining a group as the intersection, union or crossing of other groups. Implementing the corresponding algebraic constructions is standard engineering work. This would allow representing dynamic mechanisms such as "a country-theme set is made up of students such that the 'from a same country' and the 'worked at a same theme' conditions hold".

As a way to show how the T^2 model allows by-intension descriptions and configurations, we have developed an extension of the ediT2 editor to implement one of the different options for grouping students and distributing resources in the case of the jigsaw script pattern. The configuration is defined by (1) the list of students, (2) the list of topics (e.g., two topics: "insulation" and "heating"), (3) the list of resources tagged with their related topic (i.e., an indication which documentary resources address "insulation" and which address "heating"), and (4) a set of parameters. These parameters include (i) the required number of students per group (which indirectly defines the number of groups), and how to manage odd cases, and (ii) the way resources should be distributed. As examples of options for managing group odd cases: creating extended groups or an additional small group. As examples of options for distributing resources: provide all participants with all resources related to their topic or spread resources over the participants. Given the set configuration, a specific algorithm generates a tree matching the constraints (different solutions may be available): it first associates expert groups with topics (e.g., for 5 groups and 2 topics, 3 of them will work with one topic, and the other 2 with the other one) and, then, from the expert groups, (1) distributes the resources to the participants for the initial phase and (2) creates the jigsaw groups. If the configuration accepts no solution, the algorithm raises the impossibility and the reason(s) for this.

If we take the case of 22 students, a target number of 4 students per group and the option of extended groups, a solution is a tree with 78 nodes (a table with 32 sub-rows). Indeed, such a description would be difficult to manage by-hand. However, once the tree is automatically generated, the table representation can be used to manage slight modifications (e.g., move a student from one group to another) before or during the session. In this case, a variety of options are open such as allowing the teacher to apply any changes or checking whether the changes applied are contrary to the jigsaw intrinsic constraints via control rules (see next Section).

It may also be noticed that, if the editor is interoperated with an enactment framework and can retrieve data related to the script enactment, associating formulas to nodes allows dynamic grouping. For instance, if the enactment framework allows retrieving which participants have completed an activity and their productions, using formulas allow definitions such as "G1 is made up of students whose answers to the quiz Q1 were correct".

Sequencing

An intrinsic limitation of the ediT2 model is the representation of complex sequencing. Tables allow representing sequences. Representation of complex sequencing mechanisms (e.g., parallelism or loops) and, more generally, intuitive representation of the dynamic dimensions of scripts require languages building on graph models and graphical interfaces (Botturi & Todd 2007).

If used for reflecting on the script only (i.e., human interpretation) and considering simple cases, the lack of sequencing expressiveness is not necessarily an issue. For instance, in the jigsaw script presented in Figure 1, the fact that the jigsaw activity takes place after the focus activity is implicit but obvious (they are in sequence, and the resources produced in the context of the latter are inputs for the former). Similarly, the fact that the focus groups (and, later on, the jigsaw groups) may work in parallel is again implicit but rather intuitive. This, however, would not work for a complex scheduling as, for example, in the script presented in (Roschelle *et al.* 2009), which involves conditions and repetitions. While it is possible to find more or less explicit and neat ways to represent repetitions or rotations, this is more difficult for conditions.

The model may be enhanced by developing an additional mini-language to represent sequencing mechanisms and attaching these constructions to the different script-components. However, this goes against the approach rationale of exploiting the intuitiveness of table structures, and, also, would remain limited.

Another way to enhance the model, more in line with the overall approach, is to acknowledge that tables are rather pertinent to capture static descriptions, graphs are rather pertinent to capture dynamic dimensions, and both can be used when needed. Viewed in this way, enhancing the approach to represent complex scheduling mechanisms consists in connecting the T^2 representation with a graph representation allowing capturing dynamic dimensions.

As a proof of concept, we have designed and implemented a piece of code that, focusing on the Activity-Group-Participant-Resource script-structure, transforms a T^2 script into a statechart skeleton. The basic principle is to map the script notions (activities, etc.) onto the workflow concepts (we used the classical workflow engine Drools Flow as a

target (Drools 2012)). Given a script represented as a T^2 table, a specific piece of code generates a Drools Flow representation, i.e., a list of workflow components not yet connected to each other. This file can then be opened with the workflow editor, which offers specific constructions to connect the components as required and represent complex sequencing such as conditions, loops or iterations through split/join and for-each connectors.

The advantage of such an interoperation is to offer both (1) a specific editor providing easy and flexible representation of some aspects of the scripts and (2) another specific tool for representing and implementing complex sequencing (the Drools Flow interface is specifically designed to support the representation of such mechanisms, which can be defined via mouse manipulations). Within an engineering perspective, using a variety of languages may also be considered as a way to elaborate and prototype with some tools and specify with others.

3.8. Enhancing semantics

We shall now see how the model semantics may be enhanced.

We have mentioned that, basically, the model introduces a table structure. However, this table is not manipulated free of any constraint. Each manipulation is interpreted with respect to the tree representation, and accepted or not according to the constraints defined (basically, the tree constraint). This implementation approach allows enhancing the structural semantics by adding an analysis of the users' actions with respect to other constraints. Technically, this may be implemented by implementing constraints as control rules, and firing these rules when end-users act on the table.

Control rules may be used to limit model flexibility. For instance, a control rule may be used to impose a particular -or part of a particular- script-structure, relationships between notions (e.g., regulate the way Roles may be related to Groups or Participants) or relationships between items (e.g., preclude groups made up of a single participant or other odd constructions). Control rules may also be used to implement specific intrinsic constraints as defined by Dillenbourg & Tchounikine (2007), i.e., semantics related to the script rationale. For instance, if specifically considering jigsaw scripts, a control rule may be defined to check that "participants of the jigsaw groups have been in different expert groups". From a technical point of view, implementing such rules is made easy by the fact that they operate on trees, a type of algorithmic for which lambda-calculus based languages are particularly suited and efficient.

It should be noticed that, here, we only describe how the limitations of the basic semantics introduced by the table structure may be overcome, as a way to offer teachers some additional conceptual support if/when this is considered. However, regarding the issue of supporting teachers as such, many other aspects may be considered, e.g., offering T^2 skeletons of classical scripts (with, possibly, control rules related to their intrinsic constraints) or a repository of elements that may be used to describe activities or roles such as "synthesize" or "discuss" (with, possibly, control rules to check whether their use is coherent); and, of course, methodological hints.

3.9. Configuring an enactment framework

We shall finally see how the T^2 model can be used to configure an enactment framework.

One approach is to implement specific pieces of code addressing the following two aspects. First, initialize the ediT2 "resources box" with the labels of the resources previously defined in (for instance) Moodle, which then allows to use ediT2 to represent/edit the script, dragging and dropping these resource labels in the table. Second, interpret the different script-components and generate the relevant data in the Moodle database.

Another approach is to use an architecture such as Glue!PS, which allows mapping a learning design onto different frameworks (Prieto *et al.* 2011). Using this architecture only requires creating an adapter ediT2 notions / Glue!PS notions. On the enactment framework side, Glue!PS already proposes an adapter to Moodle. Using Glue!PS, a jigsaw script as presented in the preceding sections will be converted in a set of Moodle activities, offering the different resources (documents, quizzes or wikis) to the students according to the activities and groups. A central advantage is that Glue!PS allows mappings on different other frameworks (e.g., LAMS), with different modalities.

4. Validation of ease of table-tree representation and perceived flexibility

In this Section we consider the validation criterion: is the model/system easy to use by teachers?[4]

4.1. Usability test

We tested the prototypical case we had defined, namely a teacher considering an existing script (those presented in Figure 2) and editing it. The panel included five primary or secondary school teachers. We also involved one learning-scenario modeling specialist (an instructional designer) and one modeling specialist (a computer science university professor) as a way to get some possibly different input.

[4] Technical frameworks are to be measured with respect to their design rationale and specifications, i.e., in this case, ease of use. They are not to be analyzed in terms of the quality of the produced scripts or learning outcomes, since representations or operationalization frameworks may be used to implement a broad variety of collaboration scripts, including ineffective or detrimental ones (Wecker *et al.* 2010).

Script#1: a reciprocal teaching script, adapted from (Palincsar & Brown 1984)

The setting involves three or four students who are going to work on a section of text (e.g., a chapter). First, the teacher introduces different reading strategies: questioning, clarifying, summarizing and predicting. The students are required to read the text. Afterwards, one of them, acting as a teacher, lists a certain number of questions to be considered. The students discuss these questions and possibly raise others. Afterwards, the student in the teacher's role proposes an abstract; the group discusses it and modifies it until agreeing on it. Finally, the students make some predictions as to what will happen in the subsequent stages of the text. [The scenario then continues with another section text and another student acting as the teacher]

Script#2: a jigsaw script, instantiated on an energy saving topic

The scenario involves a group of four students. At the end of the process, they must produce a common document answering a set of questions. The four students are first given a common text introducing general principles related to energy saving. Then, two of the students are given a text focusing on insulation, and must produce a text listing a variety of possible techniques. The two other students are given a text focusing on heating and, yet again, must produce a text listing a variety of possible techniques. The students are then put into pairs composed of one student that worked on heating and another who worked on insulation. They are given the document listing the insulation techniques and another document listing a set of questions, and together must write a document answering these questions. Similarly, the two other students are given the document listing the heating techniques and the same set of questions, and together must write a document answering these questions. Finally, the four students are grouped. They must compare their two lists of answers and prepare a final team answer.

Figure 2. The scripts used for the usability test

The protocol is summarized in Table 5. The study was conducted to analyze trends reflecting the editor's (and underlying model's) usability and, more precisely, whether teachers succeeded in using the editor to model the script the way they wanted. The editor was introduced within its basic implementation and free of any methodological training. We did not consider if the different representations produced by the different teachers were or were not semantically equivalent to each other (or to the canonical patterns) as this is not one of the goals of the basic model/editor. We introduced two scenarios to limit the intrinsic bias that consists in introducing a computer-based tool and, within the same session, analyzing how users use it. Similarly, in the questionnaire, we introduced questions on the script and the notions (activity, role, etc.) to make sure teachers dissociated the editor usability (which is what we were interested in) from their personal perspectives or potential difficulties with the notion of script, the two scripts used as case studies and/or the notions available (however, none of the teachers had difficulties with any of these issues).

Table 5. The protocol

Phase	Content
Phase 1	The teacher was presented with a demonstration of ediT2 (basic model/interface).
Phase 2	The teacher was presented with the narrative of a first script (the reciprocal teaching script as presented in Figure 2) and asked to create a representation with the editor. The teachers were prompted as follows: "You plan to implement the following script in your classroom. Use the editor provided to create the synthetic representation you find most suited to plan the different steps and, while the script unfolds, annotate the plan, if necessary, to monitor what is happening or adapt the script".
Phase 3	A first questionnaire and debriefing were conducted to collect the teacher's first impressions, and respond to any questions related to the editor. It determined whether users were at ease with the editor features and opened up a first general discussion. Sample question: "What problems did you experience, if any?"
Phase 4	The teacher was presented with the narrative of a second script (the jigsaw script as presented in Figure 2) and asked to create a representation with the editor (prompted as in Phase 2).
Phase 5	The teacher was presented with two events related to his/her jigsaw script representation and asked to explain how he/she would react and adapt the script representation. The first event involves one student stating that he/she does not want to work with his/her assigned partner in one of the pairs defined by the teacher. The second event involves one student finishing long before the others, while working in parallel.
Phase 6	The user was presented with a final questionnaire presenting four parts. First, questions on the teachers' perspective on scripts in general, and on the notions provided by the editor to represent scripts (sample question: "I have difficulties thinking with the notions provided"). Second, questions on the editor. Third, questions on the way teachers engaged in the process and used the system (sample questions: "I found that the capacity to adapt the representation […] allowed me to reflect on the script, to refine my vision"; "When I represented the first script, I got right into it and adapted things little by little"). Finally, teachers were asked to highlight any comments or suggestions (open discussion).

The answers to the questions confirm that the editor and underlying model are intuitive and easy to use. All five teachers agreed or strongly agreed that they found the tool easy to use for the second script (four of them agreed or strongly agreed about the first script) and felt that, if they had to use the tool for a third time, it would be easy. All teachers managed the technical dimension of the interface (i.e., using drag and drop, split or merge features) almost immediately. None of them reported conceptual difficulties. When required, overcoming the 1-n relationship constraint by duplicating items was mentioned neither as a conceptual nor as a manipulation issue.

As contextual information, Table 6 presents the different script-structures defined and used by the teachers. The teachers represented the scripts as sequences of phases, where these phases, however, are conceptually addressed in different ways (see the variety of script-structures). Different conclusions may be drawn from this diversity according to matters of concern and the perspective on legitimate variations of scripts. Within the open perspective to script editing we consider, such diversity shows that the editor allows different modeling perspectives and that teachers avail themselves of the flexibility provided. If considering that instructional languages and editors should lead (support,

constrain) teachers to adopt a particular modeling perspective, such diversity shows that the basic model must be accompanied by methodological training and/or enhanced with control rules (or that another system, natively considering these goals, should be used).

Table 6. The script-structures adopted by the teachers

	Script#1	Script#2
teacher#1	Group-Role–Participant–Resource-Activity	Group-Participant-Activity-Resource
teacher#2	Participant-Group-Activity- Resource-Role	Participant-Activity-Resource-Role
teacher#3	Group-Activity-Participant-Resource-Role	Participant-Group-Resource-Activity
teacher#4	Activity-Group-Participant-Resource-Role	Activity-Group-Participant-Resource
teacher#5	Activity-Participant-Resource-Role	Activity-Participant-Resource-Role

Within our perspective, we consider as very positive the fact that all teachers mentioned their high appreciation that the provided editor was easy to use and customizable. This allowed them to keep to their usual practices while improving on them (instead of imposing another way of working). Several teachers suggested extensions such as an additional column to mention activity length or personal notes concerning how the script unfolded for adaptation in future sessions, i.e., suggestions to customize the editor according to their personal perspective and practices. This again suggests the interest of flexibility for appropriation. As an anecdote, during modeling, one teacher attempted to merge two cells when it was not possible given the 1-n constraint. She realized (after a few seconds) she could easily and neatly obtain a representation equivalent to the one she wanted by changing the column order. However, she explicitly decided not to do so, preferring to keep two separate cells with duplicate values, stating that "she preferred viewing the script this way" (i.e., with this script-structure). This suggests it is important for teachers to be able to keep to their precise view (and the corresponding script-structure).

It should be noticed that the fact teachers used different script-structures does not imply that their representations are semantically different (different representations may be semantically equivalent). A study into why teachers adopt one or another modeling perspective is most certainly an interesting topic for future research.

Although no general conclusion can be drawn given the limited number of participants, the difference between the input originating from teachers (our target users) and the two modeling specialists is worthy of note. Unsurprisingly, the computer scientist raised the issue that such a modeling tool did not support the modeler by imposing precise rules. She was not at ease with the fact that she could represent the scripts in different ways. Interestingly, the instructional designer found the tool easy to use for the first script and less so for the second (neither agrees nor disagrees). The debriefing discussion revealed that this answer pointed to difficulties with the model's expressiveness (she wanted to represent the jigsaw scheduling in a more explicit way than the basic model allows). If confirmed by further experiments, this substantiates the importance of developing and offering different representation languages and tools related to different users, matters of concern and objectives.

4.2. Analysis of the technical skills required to use the editor with respect to teachers' ICT skills

Another more general way to phrase the usability question is to consider the technical skills required to use the model as implemented by the editor, and analyze them with respect to teachers' ICT skills.

The technical skills required to use the editor are those of a table editor in a word processing office tool: add/remove rows or columns, split or merge cells, displace an element, and copy/paste. All these actions correspond to mouse manipulations (left-click, right-click, drag-and-drop).

According to a recent report on international experiences with educational technology in countries with high-performing education systems (Bakia *et al.* 2011), teachers' ICT skills are assessed in a few countries only, and there is a lack of general data. However, this report highlights that most countries consider the development of standards for teachers' ICT skills as a national priority. Online professional development for pre- and in-service teachers is available in almost half of participating countries, and many countries provide formal online or blended courses to either build teachers' capacity to integrate ICT or for more general teacher training, so as to make ICT skills an element of teacher licensing requirements. In one investigation into ICT knowledge and skill levels among Western Australian government school teachers (Trimmer 2006), word processing was part of the basic suite of ICT applications used by more than 95% of teachers. The ICT skill item map, constructed from the analysis of teachers' skills, led to a three-score division: skill scores between 0 and 39.9 (22% of teachers who typically have basic skills such as word processing and Internet), scores between 39.9 and 60.6 (53% of teachers with more advanced skills), over 60.6 (25% of teachers with even more advanced skills). The "creating tables" skill is in the middle of stage 1.

These elements suggest that a model/editor requiring the technical skills of an office table-editor will, in all likelihood, be usable by a large set of teachers. In our study, all five teachers had limited or good text editor knowledge.

It should be noticed that advocating the interest of using tables does not mean that graph-based representations are necessarily an issue. Graph representations may also be considered as intuitive. However, utilization of a workflow-like editor is not a basic ICT skill, and requires some training. More generally, work related to IMS-LD highlighted that using representations inspired from data or process modeling such as XML-like trees or process charts, which are not widespread amongst teachers, may be an issue for adoption (Neumann *et al.* 2010).

5. Discussion

Languages and editors are tradeoffs, design decisions relating to matters of concern, targeted users and/or expected utilizations. The approach presented in this article is not proposed to replace more classical graph-based and more semantically-supporting approaches but as an alternative, featuring simplicity and flexibility, to be used if and when pertinent, or in addition. In this Section, we discuss its main characteristics.

5.1. Relation to the overall operationalization issue

The top matter of concern of the approach we propose is the editing phase. Although goals are very different, this is a positioning similar to a language such as IMS-LD (IMS-LD 2012) or Collage (Hernández-Leo *et al*. 2006): the model is primarily studied to offer a certain number of features considered of importance for allowing teachers to work on the script (conceptualize, represent, edit). This option is different from building the model on-the-top of a computable representation (e.g., workflow or Petri-nets) or a particular enactment framework specification.

From a general perspective, the rationale for adopting this approach is derived from the following analysis. Without misunderstanding the interest of implementing CSCL settings via specific operationalization languages or platforms, this is far from being the most common approach in basic practices, and may not be the pattern that will be generalized. Currently, the spread of ICT technologies allows easy implementation of CSCL settings as mash-ups of off-the-shelf tools (e.g., freely downloadable and interoperable communication tools or resource-sharing tools) or local adaptations of generic platforms such as LMS, thus enabling users to mobilize technologies according to their perspectives and contexts (Tchounikine 2011). Some works propose to describe scripts with respect to the generic functionalities of Web platforms, as a strategy to allow them to be run within different enactment frameworks (Wecker *et al*. 2010). We have mentioned the Glue!PS approach, which uses a pivotal data model to deploy learning designs expressed in several design languages to different enactment frameworks (Prieto *et al*. 2011). The orchestration perspective (Dillenbourg *et al*. 2011) emphasizes the fact that, besides the technical infrastructure, efforts should focus on how to empower teachers in setting up and maintaining the learning setting. It thus makes sense to explore means to (1) support teachers in reflecting on the script and adapting it to their context, their perspective and practices (thus likely to facilitate appropriation), and (2) can be interoperated with complementary representations means and enactment frameworks.

5.2. Expressiveness and limitations

Just like any other model, T^2 captures some dimensions and not others. As it is a fairly general model, its limits relate to the level of detail of the representation rather than to the scripts that can be represented. To test the model we collected 25 macro-scripts from the literature. All of them could be represented. Nevertheless, what was captured was the dimension of the script made explicit by the T^2 model, i.e., by a perspective on scripts as a set of components (typically, phases). This corresponded to the core principle of some scripts, but missed that of scripts based on complex scheduling (e.g., based on loops or complex rotations) or enactment frameworks reifying specific constraints.

An important difference with other representation propositions such as Miao *et al*. (2005) or Kobbe *et al*. (2007) is that the T^2 model does not introduce a conceptual proposition: its goal is not to present new concepts or yet another conceptual perspective on script. The goal is of a pragmatic nature (allowing easy editing and adaptation by teachers within an open perspective to legitimate variations), and the entry point is the exploration of an idea (using tables as simple and intuitive interfaces, similar to what teachers use via office suites, and structural semantics). Model expressiveness limitations are related to this design rationale.

We have already mentioned the three implications of the model which may be considered as limitations when comparing an editor such as ediT2 with other works. First, there is a very basic representation of the dynamic dimension of scripts. In Section 3.7 we showed how a table and a graph-based representation may be combined to solve this if it appears to be an issue[5] (when considering modeling languages, it is standard practice to offer different representations capturing static and dynamic dimensions). Second and third, there is an absence of notion-to-notion specific relationships as classically described via a meta-model, and a basic semantics carried out by the label expressiveness and the table/tree structure. In Section 3.8 we explained how semantics may be enhanced by control rules in order to address such issues if/when needed.

The rationale for the T^2 model is to enlarge the diversity of means that teachers may be offered and select according to perspectives or contexts, and not to mimic already existing languages via different means. As mentioned earlier, it makes little sense to use this approach if the aspects mentioned above are considered intrinsic issues: offering powerful expressiveness and precise semantics via a meta-model is standard computer science technique, can be more easily and straightforwardly implemented, and ensures that the meta-model is respected. Representing semantics by control rules is more flexible but raises the difficulty of defining a consistent and complete set of rules. Similarly, attaching complex constructions to cells/nodes for representing (for instance) complex groupings is technically standard engineering work,

[5] As a matter of fact, when analyzing these 25 scripts collected from the literature, we noticed that complex mechanism scheduling was not that frequent and, in most cases, could easily be implemented by duplicating a limited number of rows or items. This is in line with the conclusions of Haake and his colleague who developed a language (based on Petri nets) allowing management of complex mechanisms, but noticed that in many cases simple sequences are sufficient (Haake & Pfister 2007).

but may lead to a drift from the simple design rationale. However, expressing complex by-intention mechanisms may more often be an instructional designer's concern than a teacher's concern.

We have mentioned that, from a meta-modeling perspective, the table structure may be seen as an easy-to-use way to offer end-users (very limited) meta-modeling means. From a computer science perspective, the flexibility offered by the T^2 model could be addressed by proper meta-modeling techniques. For instance, the T^2 implicit meta-model could be made explicit and represented within the system (i.e., turning the editor into a generic but instantiated editor). This would allow users not only to define the notions that can be used themselves but, also, to define notion-to-notion relationships, thus offering flexibility and (more) expressiveness and/or semantics. This meta-modeling work could be addressed partly by an instructional designer (or other modeling specialist), in the context of a project or a particular institutional setting for example, and partly by teachers, at run time. This type of model-driven engineering technique is pretty standard and has already been used for instructional languages (see for example Nodenot *et al*. 2008). Continuing in this direction, users could be presented with several representation means simultaneously modeling the script from different perspectives (see Lonchamp 2006 for example). However, these directions have already been explored. The work presented in this article explores another perspective, building on different premises.

As a matter of fact, the usability test suggests that the limitations recapped here were not to be perceived as such by the target users (to be confirmed by further experiments). Indeed, if the editor is considered from an instructional designer or a modeling specialist point of view, the implication of the ediT2 design rationale may be considered a serious issue that could disorient the user and render difficult some tasks. However, none of the tested teachers expressed such an opinion. In direct contrast, they all raised the interest of the system as a way to improve their practice. As mentioned in a recent article analyzing LD usage, with respect to scenario representation means, issues such as intrinsic complexity of languages, their appropriateness to "basic" settings, user training, user willingness, and institutional dimensions are difficult to disentangle (Derntl *et al.* 2011). Our argument here is not that teachers should be offered poor conceptual means (or that their skills should be underestimated), but that the consideration of effective practices and (lack of) training possibilities is also an important aspect. Offering simple though somewhat limited means is of core importance. This must naturally be considered within the perspective of offering teachers (and instructional designers) a variety of means.

5.3. Permissiveness

In Section 2, we introduced the notion of legitimate variations of scripts, and indicated it may be addressed with different perspectives. In fact, perspectives may be radically conflicting. We have already mentioned that any general language allows ineffective or even detrimental scripts to be represented (Wecker *et al.* 2010). However, if script edition is considered as a phase following on from script design, it could then be said that representation means should support or even oblige teachers to keep to the exact script or, at the least, close to it. The work presented in this article does not natively implement such a constraint: it empowers teachers to edit the script as they want. Natively, the T^2 model is fully permissive. Given (let's say) a representation of a canonical jigsaw script, it allows the user to change any element. The result may be totally different from a jigsaw script.

If our approach does not consider "constraining" edition according to the canonical script as being a first class objective, it is not incoherent with introducing such control/support. We indeed agree that it makes sense to support teachers in using and ensuring coherence with scripts the efficiency of which has been demonstrated by research evidence. The notion of intrinsic/extrinsic constraints is an interesting basis from which to address this issue. However, we believe this must be addressed within a pragmatic approach, balancing the "conformity to the canonical script" concern and other concerns such as adequacy to teachers' effective contexts and professional practices (conditions, training, etc.) or appropriation aspects. Moreover, the way the technology may be used to support/oblige teachers to keep close to a canonical script cannot be addressed separately from methodological training. With respect to these issues, the fact that this approach allows tuning the editor constraints may be used to investigate different balances.

5.4. Management of the table-tree representation and interface variants

Strictly imposing the 1-n with $n \geq 1$ relationship as implemented in ediT2 is a straightforward design decision that avoids ambiguous constructions while keeping the interface simple. Other options are possible. Many constructions, though not respecting this constraint and formally corresponding to graphs, are unambiguous. Some syntactical sugar could also be used to let users specify specific intentions. An interface could thus be offered that does not strictly impose this constraint but, rather, allows cells to have different antecedents as long as the construction is interpretable (keeping the machine representation as a tree to benefit from tree manipulation algorithms[6]). We have opted to keep the representation orthogonal and simple because we do not see this constraint as an important issue. Representing a script within the T^2 model does not require and does not suggest representing graphs: the model introduces a perspective on scripts as sets of components, and graphs only appear as a way to factor values. Representing this aspect via duplications is not incoherent with this model general perspective and requires little work as they can easily be managed

[6] Such graphs, if any, denote the fact that branches have common values, and can be deployed as trees via automated duplications. This is not to be confused with using a graph modeling to represent cycles as in a workflow representation.

by copy-paste and drag and drop manipulations. Viewing scripts as graphs in the full sense (i.e., to denote cyclic aspects) is a different goal, and requires different means.

More generally, in ediT2 implementation, we have kept to the model basic principles. Other options are possible. For instance, though not implementing a full-fledged meta-model approach, additional flexibility may be provided by offering the possibility to add additional columns (e.g., a "time span" column or a "personal commentary" column, as suggested by some teachers during the usability test). Another example is to allow recursive use of the notions available, e.g., decomposing activities into sub-activities. As a last example, the editor may be extended to represent information related to run-time management, e.g., the fact that an activity can be launched although some of the students are still engaged in another preceding activity or should be postponed, or the fact that an activity may be skipped or not under certain circumstances. Different interface decisions may also be made. For instance, in our implementation, splitting a cell corresponds to the creation of sub-rows (horizontal split). Vertical split could be considered as a way of adding a column at the point where the cell is split. Our design decision is that since columns denote the script's structuring notions and that the decision to use a new notion affects the entire table[7], insertion of a column is not proposed at a local level (when considering a cell) but at a global level (when considering the table). Columns are inserted to the right of the table and can then be moved to the left.

The interest of the proposed model and editor is that they are simple. Many smart ideas could be added. However, adding extensions out of any precise context is basic engineering work that could lead to complex and labyrinthine interfaces. Although computer scientists often tend to allow all possibilities at the cost of complexity, our opinion is that this is not necessarily very positive for the effective use of software in education. We rather advocate that if, in a given context (e.g., an institution or a project), the current editor principles appear pertinent but the present implementation presents hindering limitations, the latter should be adapted according to this context's specificities.

6. Conclusions and perspectives

The work presented in this article is a contribution to the on-going effort of the CSCL community to study script operationalization. It focuses on the editing issue and explores an alternative to graph-based representations, featuring simplicity and flexibility. The model is based on a straightforward visualization as a table, which allows adaptations in the form of direct manipulations. The basic expressiveness and semantics present some limitations but may be enhanced. The usability test suggests that the model/interface is easy to use and that teachers avail themselves of the flexibility available to model scripts according to their perspective.

This work raises a certain number of research questions, which we will phrase in a general way as possible perspectives for research by the CSCL community studying script operationalization.

A first set of research questions relates to how, why, and to what extent teachers use flexibility features, and in particular representation flexibility, in both lab and basic professional contexts. When teachers use this flexibility, it should be determined whether this has an impact on their appropriation of the means available (e.g., the editor available or predefined script patterns), their effective use of scripts in basic practices, and the "quality" of the represented scripts with respect to (1) the scripts' intrinsic principles (see notion of legitimate variation) and (2) the effective setting. The analysis of the quality of the produced scripts will also be a context to further investigate if the editor presents any hidden difficulty and, in such a case, what support may be offered.

A second set of research questions relates to the balance between flexibility and support, and the extent to which it may be advantageous to enhance the basic model in a given context (e.g., a project or a group of teachers), or at some point in the modeling process. A related issue is to study the implementation and interest of intrinsic and extrinsic constraints as defined in (Dillenbourg & Tchounikine 2007).

A third set of questions relates to the use of such a script representation to support a teacher-centered orchestration of the setting. In particular, one perspective is to study how a table representation may be enhanced to denote some aspects of script unfolding and be used as a monitoring device.

Finally, on a more technological note, there are research questions related to interoperation on T^2 based editors with other representational languages and technological frameworks.

7. References

Aronson, E., Blaney, N., Sikes, J., Stephan, G., & Snapp, M. (1978). *The Jigsaw Classroom*. Beverly Hills, CA: Sage Publication.

Baker, M. & Lund, K. (1997). *Promoting reflective interactions in a CSCL environment*. Journal of Computer Assisted Learning, 13(3), 175–193.

Bakia, M., Murphy, R., Anderson, K. & Trinidad, G.E., (2011). *International Experiences With Technology in Education: Final Report*. U.S. Department of Education, Office of Educational Technology. Washington, D.C.

Belgiorno, F., Chiara, R.D., Manno, I. & Scarano, V. (2008). *A Flexible and Tailorable Architecture for Scripts in F2F Collaboration*. In Proceedings of the 3rd European Conference on Technology Enhanced Learning: Times of Convergence:

[7] Creating a column limited to some of the rows is not in the model's scope as it does not comply with the homogeneity principle and the interface principle.

Technologies Across Learning Contexts. Springer-Verlag Berlin, pp. 401-412.

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

Botturi, L., Derntl, M., Boot, E. & Figl, K. (2006). *A Classification Framework for Educational Modeling Languages in Instructional Design*. In Proceedings of 6th IEEE International Conference on Advanced Learning Technologies, pp. 1216-1220.

Derntl, M., Neumann, S., Griffiths D. & Oberhuemer, P. (2011). *The conceptual structure of IMS Learning Design does not impede its use for authoring*. IEEE Transactions on Learning Technologies (preprint). http://dx.doi.org/10.1109/TLT.2011.25

De Wever, B., Van Keer, H., Schellens, T., & Valcke, M. (2009). *Structuring asynchronous discussion groups: The impact of role assignment and self-assessment on students' levels of knowledge construction through social negotiation*. Journal of Computer Assisted Learning, 25, 177–188.

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

Dillenbourg, P., Dimitriadis, Y., Nussbaum, M., Prieto, L.P., Asensio, J.I., Sharples, M., Fischer, F., Kollar, I., Tchounikine, P., Roschelle, J., Looi, C.K., Diaz, A., Hämäläinen, R., De Wever, B., Perrotta, C., Evans, M.A., Chan, T.W. & Balaam, M. (2011). *Classroom Orchestration* (submitted; extended version of Deliverable D1.5 of the European Network for Excellence Stellar).

Dillenbourg, P. & Hong, F. (2008). *The mechanics of CSCL macro scripts*. International Journal of Computer-Supported Collaborative Learning, Springer 3 (1), 5-23.

Dillenbourg, P. & Jermann, P. (2007). *Designing Integrative Scripts*. F. Fischer, I. Kollar, H. Mandl & J. M. Haake (Eds.). Computer-Supported Collaborative Learning Series, Springer, pp.275-301.

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

Drools. *The Business Logic integration Platform* (www.jboss.org/drools), visited in 08/2012.

Fischer, F., Kollar, I., Haake, J.M. & Mandl, H. (2007). *Perspectives on collaboration scripts*. In F. Fischer, I. Kollar, H. Mandl & J. M. Haake (Eds.), Scripting computer-supported communication of knowledge – cognitive, computational, and educational perspectives, New York: Springer, pp. 1-10.

Haake, J.M. & Pfister, H.-R. (2007). *Flexible scripting in Net-Based Learning Groups*. F. Fischer, I. Kollar, H. Mandl & J. M. Haake (Eds.). Computer-Supported Collaborative Learning Series, Springer, pp.155-175.

Harrer, A. & Hoppe, H. U. (2008). *Visual Modeling of Collaborative Learning Processes – Uses, Desired Properties and Approaches*. In L. Botturi & S. Todd (Eds.). Handbook of Visual Languages for Instructional Design: Theory and Practices, Information Science Reference, Hershey, pp. 281-298.

Harrer, A., Malzahn, N. & Hoppe, H.U. (2007). *Graphical Modeling and Simulation of Learning Designs*. In T. Hirashima, H. U. Hoppe and S. S. Young (Eds.). Supporting Learning Flow through Integrative Technologies. Amsterdam, IOS Press, pp. 291-294.

Hernández-Leo, D., Asensio-Pérez, J.I., Dimitriadis, Y., Villasclaras-Fernández, E.D. (2010). *Generating CSCL Scripts: From a Conceptual Model of Pattern Languages to the Design a real situation* (Appendix). In: Goodyear, P., Retalis, S. (Eds) Technology Enhanced Learning, Design Patterns and Pattern Languages, SensePublishers.

Hernández-Leo, D., Villasclaras-Fernández, E.D., Asensio-Pérez, J.I., Dimitriadis, Y., Jorrín-Abellán, I.M., Ruiz-Requies, I. & Rubia-Avi, B. (2006). *COLLAGE: A collaborative Learning Design editor based on patterns*. Journal of Educational Technology and Society, 9(1), 58-71.

IMS-LD. *IMS Learning Design* (http://www.imsglobal.org/learningdesign), visited 08/2012.

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.

Kollar, I., Fischer, F., & Slotta, J.D. (2007). *Internal and external scripts in computer-supported collaborative inquiry learning*. Learning and Instruction, 17(6), 708–721.

Kollar, I., Fischer, F. & Hesse, F.W. (2006). *Computer-supported cooperation scripts – a conceptual analysis*. Educational Psychology Review, 18(2), 159-185.

LAMS. *LAMS: Learning Activity Management System* (http://www.lamsfoundation.org/), visited in 08/2012.

Lonchamp, J. (2006) Supporting synchronous collaborative learning: A generic, multi-dimensional model. International Journal of Computer-Supported Collaborative Learning 1(2), 247-276.

Miao, Y., Hoeksema, K., Hoppe, H.U. & Harrer, A. (2005). *CSCL Scripts: Modelling Features and Potential Use*. International Conference on Computer Supported Collaborative Learning, Taipei, Taiwan, pp. 423-432.

Moodle. *Open-source community-based tools for learning* (www.moodle.org), visited in 08/2012.

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 (IJET), 5(1), 58-72.

Nodenot, T., Caron, P-A., Le Pallec, X., Laforcade P. (2008). *Applying Model Driven Engineering Techniques and Tools to the Planets Game Learning Scenario*. Journal of Interactive Media in Education (http://jime.open.ac.uk), Special Issue: Comparing Educational Modelling Languages on the "Planet Game" Case Study.

O'Donnell, A.M. (1999). *Structuring dyadic interaction through scripted cooperation*. In A. M. O'Donnell & A. King, eds, Cognitive perspectives on peer learning, pp. 179-196. Mahwah, NJ: Erlbaum.

Palincsar, A., & Brown, A. (1984) *Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities*. Cognition and Instruction (1), 117-175.

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.

Ronen, M., Kohen-Vacs, D. & Raz-Fogel, N. (2006). *Adopt & Adapt: Structuring, Sharing and Reusing Asynchronous Collaborative Pedagogy*. International Conference of the Learning Sciences, pp. 599-605.

Roschelle, J., Rafanan, K., Bhanot, R., Estrella, G., Penuel, B., Nussbaum M. & Claro, S. (2009). *Scaffolding group explanation and feedback with handheld technology: impact on students' mathematics learning*. Education Tech Research Dev, Springer, 58(4), pp. 399-419.

Rummel, N., & Spada, H. (2005). *Learning to collaborate: An instructional approach to promoting collaborative problem solving in computer-mediated settings*. The Journal of the Learning Sciences, 14 (2), 201–241.

Schellens, T., Van Keer, H., De Wever, B., & Valcke, M. (2007). *Scripting by assigning roles: Does it improve knowledge construction in asynchronous discussion groups?* International Journal of Computer-Supported Collaborative Learning, 2(2–3), 225–246.

Schoonenboom, J. (2008). *The effect of a script and an interface in grounding discussions*. International Journal of Computer-Supported Collaborative Learning, 3(3), 327–341.

Slof, B., Erkens, G., Kirschner, P. A., Jaspers, J.G. M., & Janssen, J. (2010). *Guiding students' online complex learning-task behavior through representational scripting*. Computers in Human Behavior, 26 (5), 927–939.

Stahl, G., & Hesse, F. (2007). *Welcome to the future: ijCSCL volume 2*. International Journal of Computer-Supported Collaborative Learning, 2(1), 1–5.

Stegmann, K., Weinberger, A., & Fischer, F. (2007). *Facilitating argumentative knowledge construction with computer-supported collaboration scripts*. International Journal of Computer-Supported Collaborative Learning, 2(4), 421–447.

Tchounikine, P. (2011). *Computer Science and Educational Software Design – A Resource for Multidisciplinary Work in Technology Enhanced Learning*. Springer. DOI 10.1007/978-3-642-20003-8_6

Tchounikine, P. (2008). *Operationalizing macro-scripts in CSCL technological settings*. International Journal of Computer-Supported Collaborative Learning, Springer, 3(2), 193-33.

Trimmer, K. (2006). *Teacher ICT Skills: Evaluation of the Information and Communication Technology Knowledge and Skill Levels of Western Australian Government School Teachers.* Autralian Evaluation Society International Conference, Darwin, Australia. (retrieved from http://www.aes.asn.au, April 2012).

Villasclaras-Fernández, E. D., Hernández-Gonzalo, J. A., Hernández-Leo, D., Asensio-Pérez, J. I., Dimitriadis, Y., Martínez-Monés, A. (2009). *InstanceCollage: a tool for the particularization of collaborative IMS-LD scripts*. Educational Technology &Society, 12 (3), 56–70.

Wecker, C., Stegmann, K., Bernstein, F., Huber, M.J., Kalus, G., Kollar, I., Rathmayer, S. & Fischer, F. (2010) *S-COL: A Copernican turn for the development of flexibly reusable collaboration scripts*. International Journal of Computer-Supported Collaborative Learning, Springer New York, 5, 321-343.

Weinberger, A., Ertl, B., Fischer, F., & Mandl, H. (2005). *Epistemic and social scripts in computer-supported collaborative learning*. Instructional Science, 33(1), 1–30.

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*: Berlin:Springer, 155-174.

Weinberger, A., Stegmann, K., & Fischer, F. (2010). *Learning to argue online: Scripted groups surpass individuals (unscripted groups do not)*. Computers in Human Behavior, 26, 506–515.

Williams, D.L., Boone, R., & Kingsley, K.V. (2004). *Teacher beliefs about educational software: A Delphi study*. Journal of Research on Technology in Education, 36(3), 213-229.