Flexibility in macro-scripts for CSCL

Pierre Dillenbourg and Pierre Tchounikine

Ecole Polytechnique Fédérale de Lausanne, Suisse; pierre.dillenbourg@epfl.ch
Université du Maine, France; pierre.tchounikine@lium.univ-lemans.fr

Abstract. In the field of computer-supported collaborative learning (CSCL), scripts are designed to support collaboration among distant learners or co-present learners whose interactions are (at least partially) mediated by a computer. The rationale of scripts is to structure collaborative learning processes in order to trigger group interactions that may be rare in free collaboration. Fixing the degree of coercion is a delicate design choice: too rigid scripts would spoil the richness of collaborative interactions; too flexible scripts would fail to induce the targeted interactions. Because of the unpredictability of how scripts will be enacted, both the teacher and the students must be allowed to modify some script features. In this article we propose a conceptual analysis of this notion of flexibility, arguing for a dissociation of constraints that are intrinsic to the pedagogical design of the script and constraints induced by the technology or contextual factors. This difference sets up the limits of flexibility both for the teacher and for the students and provides specification for the computational design. This analysis leads to the conclusion that the operationalization of CSCL scripts should be addressed by implementing script engines handling multiple representations of the script: the script to be executed, the emergent organization of teams, the set of intrinsic and extrinsic constraints and the visual representation of the script for students and teachers.

Keywords: collaborative learning, scripts, flexibility, computational model.

1 Introduction

Scripts aim at structuring collaborative processes by defining sequences of activities, by creating roles within groups and by constraining the mode of interaction among peers or between groups. (Jermann & Dillenbourg, 1999; Kollar, Fischer & Hesse, in press). Scripts originate from the fact that it is difficult to predict the effects of collaborative learning by controlling external conditions such as group composition or task features. Actually, the effects of collaborative learning depend on the quality of interactions that take place among group members. Therefore, scripts aim to enhance the probability that knowledge generative interactions such as conflict resolution, explanation or mutual regulation occur during the collaboration process. In the field of computer-supported collaborative learning (CSCL), scripts are designed both to support collaboration among distant learners and to mediate collaboration among co-present learners. In this article we address *macro-scripts*, i.e., coarse-grained scripts that aim at creating learning situations within which the desired interactions between students should occur. We use the macro-script notion as opposed to the micro-script notion, i.e., finer-grained scripts that follow a more psychological approach and emphasize the activities of individual learners. A more precise definition of macro-scripts is provided in section 2.

Using computers to run CSCL scripts has advantages such as the fact that the script can be run among distant learners, that the computer can manage complex workflows even with large groups or that the process gains in explicitness and traceability. It also has drawbacks, such as introducing constraints that are not part of the pedagogical intentions. This contribution addresses the issue of flexibility in CSCL scripts. Of course, the very idea of a script embeds a decrease of flexibility: a script aims at structuring group processes, which requires some rigidity. However, because of the unpredictability of how scripts will be enacted, both the teacher and the students must be allowed to modify some script features. What aspects of the scripts should – by design – be non-flexible and which ones should – despite implementation – remain flexible? We propose a conceptual analysis of flexibility in CSCL macro-scripts, arguing for a dissociation of constraints that are intrinsic to the pedagogical design of the script and constraints induced by the technology or contextual factors. The objective is to help designers in disentangling the flexibility loss inherent to the pedagogical intentions from the flexibility loss that is an undesired effect of translating the script idea into a computer program. This analysis allows us to draw general specifications for the design of CSCL operationalization platforms: the so called script engine must maintain the script intrinsic constraints whilst allowing some flexibility, the space for flexibility being related to the extrinsic constraints. For this purpose, the script engine must handle multiple representations of the script.

This article addresses conceptual issues rather than empirical evidence of script effectiveness. Several studies revealed that scripts are effective in case-based environment (Weinberger et al., 2005), videoconferencing tools (Rummel & Spada, 2005), discussion forums (Schellens et al., 2005) and other virtual environments (Hämäläinen et al., 2005). These studies do not conclude that, considering learning outcomes, scripts are effective in general, but that specific scripts with specific features have been effective. In our work, we collected empirical evidence for the ArgueGraph script (Jermann & Dillenbourg, 2003), but overall we experienced that the effects of a script are fragile. This does not only depend on the intrinsic quality of the scripts but also depends, to a great extent, on the way the teacher may adapt this script to the learning settings. While this is true for any piece of educational software, it is more salient for scripting environments which interplay with the class processes. The way teachers (and students) may adapt the script to their context was then identified to be a key variable to be explored in future works.

This article is organized as follows. In section 2, we present three examples of CSCL macro-scripts (§2.1.). These examples will be used all along the article to exemplify and anchor our arguments about flexibility. We then propose definitions for the macro-script notion and list macro-scripts important characteristics. In section 3, we deepen our analysis of flexibility. We first dissociate intrinsic constraints and extrinsic constraints. We then analyze what flexibility issues are required for the teacher (§3.1) and then for the students (§3.2), using the examples introduced in §2 to exemplify intrinsic and extrinsic constraints. In section 4, we define what conclusions can be drawn from this analysis in terms of operationalization platforms.

2 CSCL scripts

2.1 Examples

We present here below three examples of macro-scripts that aim at anchoring our arguments about flexibility. These scripts have been tested in authentic learning contexts (at the university level).

The ***ArgueGraph script*** (Jermann & Dillenbourg, 2003) aims at triggering argumentation among pair members. ArgueGraph includes 5 phases. 1) Each student takes a quiz on-line, this quiz being related to the studied content domain (e.g., how to react to students' erroneous answers within educational software). The multiple-choice questions have no correct or incorrect answer; students' answers reflect their opinions. For each choice, the student enters an argument in a free-text entry zone. 2) The second phase occurs face-to-face with all students. The system produces a simple graph in which students are positioned according to their answers. The students comment freely on this social map, which increases the social dynamics in the class. The teacher forms pairs by selecting students with the largest distance on the graph (i.e., with the most different opinions). 3) Pairs sit in front of a computer and answer the same questionnaire as in phase 1 together. Again, they complement their choice by an open-text argument. 4) All students meet again with the teacher who reviews the answers and arguments produced by individuals and teams. The system therefore aggregates the collected answers and the arguments that were given. As these arguments are not structured, the teacher's role is to organize them, to rephrase them with proper concepts and to articulate them with theories. This debriefing phase is the richest interaction phase but has so far been conducted only in small classes (up to 25 participants). 5) Each student individually writes a structured synthesis of the arguments collected for a specific question. This script has been operationalized by a platform proposing functionalities such as generating the graph from pairs and a tool to form pairs and compile responses. As represented on the vertical axis in Figure 1, this graph does not only include collaborative activities but integrated individual, collaborative and class-wide activity.

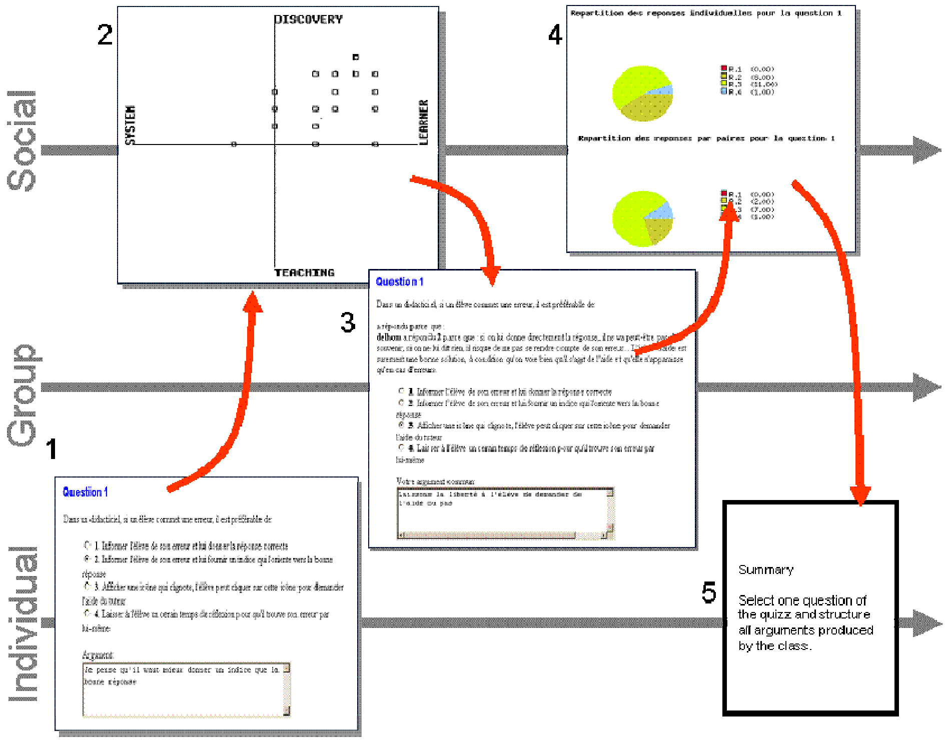
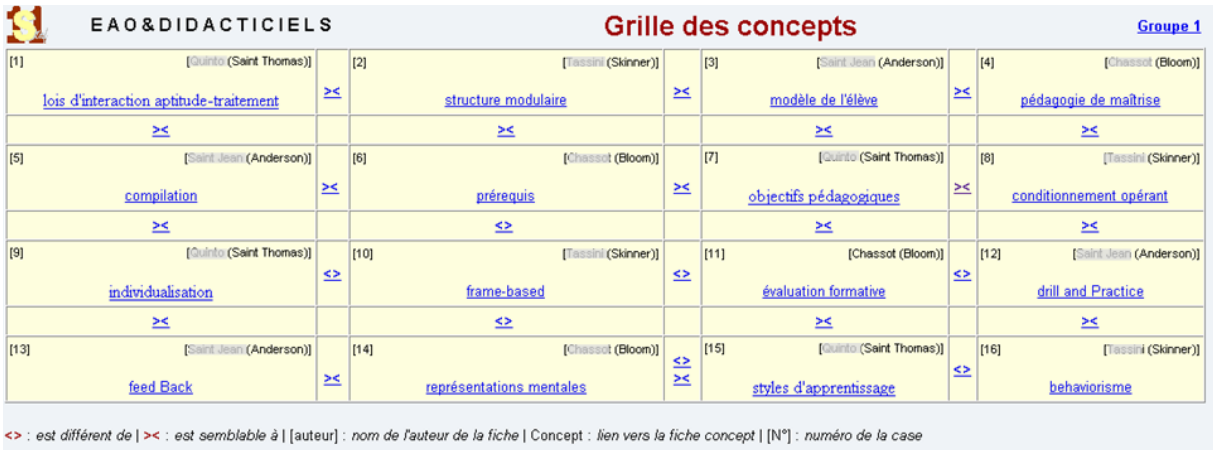

Social
Group
Individual
1
2
3
4
5
Question 1
DISCOVERY
SYSTEM
LEARNER
TEACHING
Repartition des reponses individuelles pour la question 1
Repartition des reponses par paires pour la question 1
Summary
Select one question of the quizz and structure all arguments produced by the class.

Figure 1. The ArgueGraph script: time is represented horizontally and social plane vertically; arrows represent the dataflow between activities.

The ***Concept Grid script*** (Dillenbourg, 2002) is a sub-class of the Jigsaw family, i.e., scripts that are based on making individual students manage some partial knowledge and then prompting them to solve collectively a problem that necessitates knowledge from each of them. Concept Grid includes 5 phases. 1) Groups of four students have to distribute four roles among themselves. Roles correspond to theoretical approaches of the domain under study (in our case, the domain was learning theories and roles were for instance Skinner, Bloom or Anderson). In order to learn how to play their roles, students have to read 3 papers that describe the related theory. 2) Each group receives a list of concepts to be defined and distributes these concepts among its members. Students write a 10-20 line definition of the concepts that were allocated to them. 3) Groups have to assemble these concepts into a grid and to define the relationship between grid neighbors. The key task is to write 5 lines that relate or discriminate two juxtaposed concepts: if Concept-A has been defined by Student-A and Concept-B by Student-B, writing the Concept-A/Concept-B link requires Student-A to explain Concept-A to Student-B and *vice versa*. The grid to be filled is a simple 4 X 4 html empty table as represented in Figure 2. 4) During the debriefing session, the teacher compares the grid produced by different groups and asks them to justify divergences. This script has been operationalized by a platform proposing the grid construction.

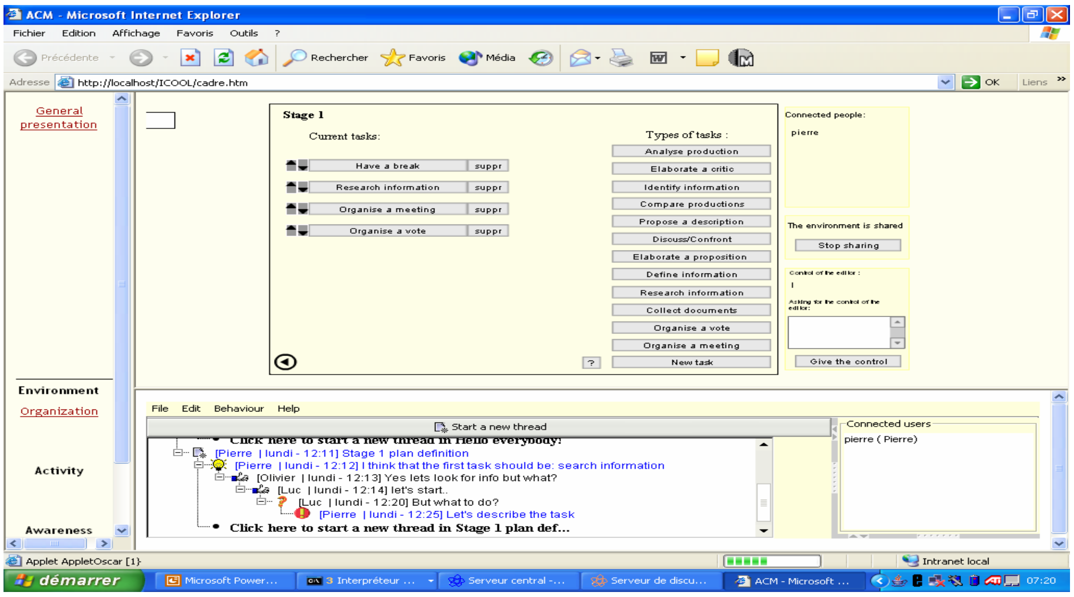
The ***RSC script*** (Betbeder & Tchounikine, 2003) is a subclass of the project-based family, i.e., scripts that are based on making a set of students work as a project team that tackles a common goal within a period of a few weeks, e.g., the elaboration of a grid of ergonomic principles for pedagogic Websites. RSC includes 3 phases (Research - Structure - Confront) which can be repeated several times, the output of a phase being the input of the next one. 1) Each student of the group has to freely research the Internet to discover some information and become familiar with a given same topic, e.g., ergonomics. 2) Each student has to structure and/or use the data he/she has recovered according to a task (e.g. elaborate a grid of ergonomic principles that allows analyzing educational Websites). 3) The group has to elaborate a collective construction from the individual productions, e.g., confront the individual productions and collectively construct a grid and then an analysis of some Websites. This coarse-grained abstract script has been used to address different objectives such as getting learners familiar with a topic (e.g., ergonomics) and also practicing organization, i.e., making students work out issues such as making the task to be achieved explicit, decomposing the task into different individual and collective subtasks, scheduling the tasks, managing time issues or identifying the resources and tools to be used to perform each task. This script has been operationalized by a platform proposing an explicit dissociation of an "organizational level" and an "activity level". The organization level allows students to conceptualize and describe their organization as a set of phases, where each phase is structured as a plan (sequence of tasks, cf. Figure 3) and each task is defined by different features such as the actors, the roles, the constraints and the resources and tools that should be available at the activity level to achieve the tasks. The activity level provides the resources and tools asked for at the organization level.


EAO&DIDACTICIELS — Grille des concepts — Groupe 1

[1] [Quinto (Saint Thomas)] lois d'interaction aptitude-traitement	><	[2] [Tassini (Skinner)] structure modulaire	><	[3] [Saint Jean (Anderson)] modèle de l'élève	><	[4] [Chassot (Bloom)] pédagogie de maîtrise
><		><		><		><
[5] [Saint Jean (Anderson)] compilation	><	[6] [Chassot (Bloom)] prérequis	><	[7] [Quinto (Saint Thomas)] objectifs pédagogiques	><	[8] [Tassini (Skinner)] conditionnement opérant
><		<>		><		><
[9] [Quinto (Saint Thomas)] individualisation	<>	[10] [Tassini (Skinner)] frame-based	<>	[11] [Chassot (Bloom)] évaluation formative	<>	[12] [Saint Jean (Anderson)] drill and Practice
><		<>		><		><
[13] [Saint Jean (Anderson)] feed Back	><	[14] [Chassot (Bloom)] représentations mentales	<> ><	[15] [Quinto (Saint Thomas)] styles d'apprentissage	<>	[16] [Tassini (Skinner)] behaviorisme

<> : *est différent de* | >< : *est semblable à* | [auteur] : *nom de l'auteur de la fiche* | Concept : *lien vers la fiche concept* | [N°] : *numéro de la case*

Figure 2. The grid that students had to construct. Blurred names are the students' names while other names are the roles to play. Students had to edit the link between the concepts of neighbour cells.



Click here to start a new thread in Hello everybody!
[Pierre | lundi - 12:11] Stage 1 plan definition
[Pierre | lundi - 12:12] I think that the first task should be: search information
[Olivier | lundi - 12:13] Yes lets look for info but what?
[Luc | lundi - 12:14] let's start..
[Luc | lundi - 12:20] But what to do?
[Pierre | lundi - 12:25] Let's describe the task
Click here to start a new thread in Stage 1 plan def...

Figure 3. Defining a plan within Symba organization editor: the upper part of the interface provides a shared editor, the lower part an on-line chat (in this case, the Oscar structured chat (Delium, 2003)).

2.2 Definitions

In cognitive science, the term *'script'* has been used to refer to the mental representation of procedures we use in everyday situations such as entering restaurant, participating in a professional meeting or walking into a plane (Schank & Abelson, 1977). The term *'script'* has also been used for describing methods that structure face-to-face collaborative learning (Aronson et al., 1978; Palincsar & Brown, 1984; O'Donnell & Dansereau, 1992). We use scripts here in the latter meaning, i.e., as a pedagogical scenario used in a computer-mediated setting.

A *macro-script* is a pedagogical method that aims at producing desired interactions. The term macro-script is used to differentiate our scripts from *micro-scripts* (Dillenbourg & Jermann, to appear). Let us illustrate the macro/micro difference with scripts that aim at raising argumentative dialogues. A macro-script sets up conditions in which argumentation should occur, as in the ArgueGraph, for instance by pairing students with opposite opinions. A micro-script scaffolds the interaction process per se: when a learner brings an argument, the script will for instance prompt his or her peer to state a counter-argument (Kollar et al, in press). These two examples differ by their granularity: a phase in a macro-script is an activity that may last for several weeks while when in a micro-script, it may be a single conversational turn. This degree of granularity is not binary; hence the micro/macro distinction could be considered as a continuum. However, these two examples have a very different status: the ArgueGraph is a pedagogical method that simply needs to be adopted by the students, while Kollar et al's scripts convey a model of dialogue that is expected to be internalized by students. Actually, a script such as RSC is 'macro' in terms of granularity but some of its features are nonetheless expected to be internalized since students are expected to internalize concepts such as “plan” or “tasks” and /or to build an explicit organization.

Macro-scripts such as the examples we have proposed include individual phases, collaborative phases (i.e., intensive interactions within small groups) and collective phases (i.e. looser interactions within larger groups such as a class-wide discussion). We refer to *integrated learning scripts* as scripts that integrate individual, collaborative and collective activities – part of which can be computerized – into a consistent pedagogical scenario (Dillenbourg & Jermann, to appear).

The *degree of coercion* of a script defines the constraints imposed by the script. Within macro-scripts, the degree of coercion can be very different. For instance, in the RSC script the group organization is only suggested and induced by the set of offered tools, while the ArgueGraph or Concept Grid are structured by explicit instructions and include a sequence of mandatory activities.

Macro-scripts can be seen as composed of a *core script*, i.e., the mechanisms by which targeted interactions occur, and a *didactic envelope*, i.e., the set of pre/post activities that allow triggering the core mechanisms (Dillenbourg & Jermann, to appear). The core mechanisms are conflict resolution for the ArgueGraph script, mutual explanation for the ConceptGrid script and confrontation and/or organization in the RSC script. The didactic envelope includes the advanced/post organizers, mostly setup at the individual and collective levels.

Finally, we need to discriminate the *script* (i.e. what is expected to be executed) and the *actual interaction pattern* (i.e., the script as it actually unfolds as a set of activities and interactions taking place among the students). For instance, in the ArgueGraph, students are paired on the basis of their global opinions (sum of 10 answers) and hence may agree on some answers: the actual degree of disagreement can be different from the expected one. In the ConceptGrid, one student may already be familiar with the theories read by another team member and hence be able to articulate some concepts on the grid without mutual explanations. In the RSC script, students may adhere to the suggested organization or organize themselves in a completely different way. Flexibility is related to this distance between the prescribed activities and the actual interaction patterns.

3 Flexibility

3.1 Why should scripts be flexible?

CSCL macro-scripts are at the junction of instructional design and sociocultural approaches. Designing and implementing such scripts requires addressing a pedagogical dilemma that is very classical but particularly salient in CSCL: if the scaffolding is too weak, it will not produce the expected interactions; if it is too strong, it will spoil the natural richness of free collaboration. The purpose of a script is to introduce constraints that will shape collaborative interactions whilst avoiding the risk of *over-scripting* collaboration (Dillenbourg, 2002), i.e., constraining collaboration in a way that makes it sterile by inhibiting the “natural” peer interaction mechanisms.

In order not to over-constrain the pedagogic setting scripts must be flexible, i.e., adaptable by the students and/or the teachers. This is in particular the case for coarse-grained scripts such as macro-scripts (globally, the degree of coercion of a script is related to its time/task granularity and scripts with a fine grained sequence are more coercitive). For instance, teachers should be able to deal with and/or take advantage of the characteristics of the way the script is enacted and students should not be unnecessarily constrained if they organize themselves in a way that slightly differs from the script. Such changes must however be analyzed with respect to the pedagogical intention: not any change should be allowed.

In order to facilitate the design and operationalization of flexible scripts avoiding the risk of over-scripting whilst respecting the pedagogical issues underlying the script, we propose to use the notions of intrinsic and extrinsic constraints.

- *Intrinsic constraints* are bound to the core mechanisms of the scripts, i.e., to the script design rationale, e.g., “each student must argue with somebody having opposite opinions” or “students must build a common text from their individual productions”.
- *Extrinsic constraints* are induced by different issues such as technological choices (e.g., using a simple html editor for the ConceptGrid script), contextual factors (e.g., the University imposes individual marks) or arbitrary decisions (e.g., while the ConceptGrid can be used with teams from 2 to 6 students, we arbitrarily fixed 4).

Extrinsic constraints define the space within which a script should be modifiable by teachers and/or students. Intrinsic constraints set up the limits of flexibility, i.e., what cannot be accepted in order for the script to keep its *raison d'être*. For instance, allowing free pairing in the ArgueGraph script would violate an intrinsic constraint and would break the link between the script and the underlying learning principle.

The intrinsic/extrinsic distinction questions design decisions: what flexibility is offered to teachers and students? This distinction is however not proposed as a way to partition design decisions (in the mathematical sense of: dissociating all design decisions in different sets). In some cases, some design decisions are undoubtedly intrinsic or extrinsic, they must be taken into consideration as such and this may provide guidelines for some further important design decisions. In some cases, some design decisions appear to be both (and more or less) intrinsic and extrinsic and/or to what extend they are intrinsic or extrinsic can appear to be of little interest. However, in all cases, questioning in a explicit way the nature of constraints is of interest, at least as a way to question the design decisions and discover implicit/uncontrolled derivations of the design process that leaded to losses of flexibility that are not related to any pedagogical issue (and, eventually, that are not related to any explicit decision).

Hereafter, we discuss the flexibility that should be offered to the teacher (§3.1) and then to the students (§3.2).

3.2 Flexibility for the teacher

When teachers choose to use a script, they have to adapt it to their context. These different modifications can be conceptualized within 4 phases. Figure 4 describes these phases in the case of the ArgueGraph script.

1. Script edition. This corresponds to the modification of the script structure to better fit the pedagogical goals of the teacher. The teacher might for instance change the order of phases, add or remove a phase, or associate a given tool to a given phase. As an example, the teacher might decide to replace the ArgueGraph free argumentation phase by the use of an argumentation tool, producing a new script referred to in Figure 4 as ArgueGraph*.
2. Script instantiation. This corresponds to "filling" the script with the relevant content. At this stage, the teacher will insert content-specific resources (documents, pictures, etc.) that will be used by the students. As an example, if the script is used within a biology course, the teacher has to specify the questions to be asked about biology, the axes for the opinions map and the textual resources. This instantiated script is referred to in Figure 4 as BioArgueGraph.
3. Session set up. This corresponds to the definition of the session parameters. At this stage the teacher will specify the group composition (or the group composition procedure) and the duration for each phase. As an example, the outcome can be the script referred to in Figure 4 as BioArgueGraph-364, which is ready to be executed on Tuesday March 24 with the students of the "Biology-101" course.
4. Run time. This corresponds to the management of the script enactment. The teacher sometimes need to modify the script structure (e.g., change the argumentation tool because students face problems with it), the content (e.g., modify a question that appears to be unclear for most students), the groups' composition (e.g., manage the fact 2 students dropped out the course) or the time schedule (e.g., extend a phase) in order to manage unexpected events.

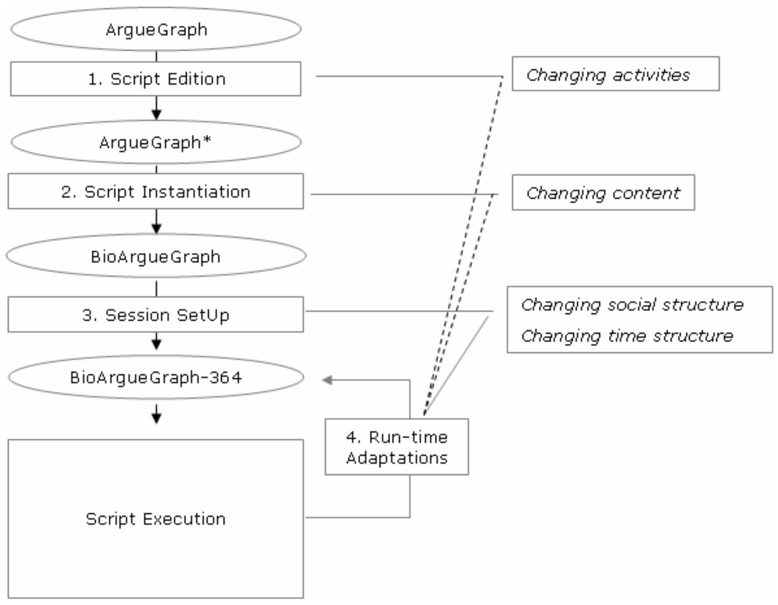


Figure 4. Times where teachers may modify a script

At each phase, the modifications desired by the teacher may be more or less acceptable with respect to the script *raison d'être*. Acceptability can be assessed by checking if the modification violates intrinsic or extrinsic constraints. Table 1 provides examples of both types of modifications for each phase. Since intrinsic/extrinsic constraints are not always clear-cut categories, we consider – for each modification – if it violates "rather intrinsic" versus "rather extrinsic" constraints.

Flexibility Points	**Intrinsic Pole**	**Extrinsic Pole**

Script edition	Changing activities or their order	In the ArgueGraph script, the teacher may suppress phase 4 (debriefing) or phase 5 (reflection) that belong to the didactic envelope of the script. He or she may not suppress phase 3 (argumentation) that is the core component of the script. In the RSC script, confrontation is triggered by the fact the group must work on the basis of the individual productions built during the preceding phase. Therefore, the order of phases cannot be altered.	In the ConceptGrid script, the teacher could replace the grid construction by a similar activity such as building a concept map. This modification concerns a core component of the script, but the fact that this core activity is performed on a grid or a graph is rather an extrinsic decision (although we could argue that a grid is more constraining than a graph).
Script instantiation	Changing contents	Scripts can be generalized but cannot be used for all contents. When using the ArgueGraph, the teacher has to produce the questions necessary to capture conflicting opinions and provide a graph [x y] value for each answer. The intrinsic constraint lies in the fact that the contents must be arguable and can be segmented into a discrete set of questions. In the RSC script, the intrinsic constraint is that the task must be sufficiently complex to require organization skills.	Some extrinsic constraints lie in the teachers' interface for entering the content. In the current version of ArgueGraph, the questions do for instance not include any picture. Hence, for arbitrary design choices, we could not use ArgueGraph for X-ray interpretations or any similar argument that relies on pictures. When the RSC script is used to make students work on organizational issues, the domain that is addressed by the activity (i.e., the fact the students address ergonomics or biology related issues) is a free variable and can be linked to extrinsic constraints (relations with a course of the curriculum, etc.).
Session set up	Defining groups	The ideal group size is respectively 2 for the ArgueGraph, 3-5 for the ConceptGrid and 6-8 for the RSC script. This optimal size depends on the targeted interactions. Trying to run these scripts with teams of respectively 6, 10 and 30 students would certainly be a failure	A former version of ConceptGrid ran with teams of 4, simply because we did not implement group size as a variable. A team of 3, that is pedagogically acceptable, was not possible because of constraints described in section 4.2.
	Defining time frame	We found that the ArgueGraph debriefing phase has to come immediately after the argumentation phase, otherwise the social dynamics created by the argumentation phase disappears. In RSC, the project must go on for several weeks, which requires organization to be explicitly worked out (defining a plan, comparing with effective actions, revising, etc.). Attempting to run the RSC script over a 60 minute task might hence not be appropriate	Most deadlines result from extrinsic decisions. If the ConceptGrid designer gives to students 10 days for reading the papers, this is a rather arbitrary choice that the teachers should be allowed to modify.

Run time	Changing groups	Running scripts often require changing groups, for instance because a member drops out of the course in the middle of the script or two conflicting leaders emerge in a RSC group and attempt to rule it from their opposing points of view. Adding, removing or moving students to/from groups may violate intrinsic constraints. For instance, in Concept Grid, changes to the group composition are constrained by the fact that students should have different knowledge.	Sometimes, adding/removing a group member may be impossible for computational reasons, i.e., because the developer did not anticipate that some data could be missing (see 4.2).
	Changing deadlines	In the ArgueGraph, postponing phase 1 would spoil phase 2 (which is not the case in the ConceptGrid). It often occurs that script deadlines have to be postponed for external reasons (e.g., network failure) or for all reasons that students invent but in the RSC script a teacher may refuse to postpone deadlines if he/she believes the organization necessary to stick to deadlines is part of the objectives.	In the RSC script the fact that a significant number of students have proposed individual productions is required for the collective phase to begin, but it can be admitted that some students send their productions later on: it is not a dead-lock if a few students do not send anything as the group can continue without them and/or they can join the group whilst not having achieved their individual job.

Table 1. Intrinsic and extrinsic constraints on macro-scripts flexibility

These points illustrate why flexibility is a more complex issue in CSCL scripts than in standard authoring tools. First, the script design encompasses a pedagogical intention (intrinsic constraints) that script changes could break down, making the script meaningless. Second, groups are unpredictable entities, even more difficult to predict than individual learners; what happens may make it necessary for the teacher to modify the script at any time. Hence, a major feature of integrated scripts is that the teacher remains the orchestra-conductor, orchestrating the sequence of activities. When the teacher is analyzing the script as it is enacted by the students, he/she eventually has to manage unexpected events (originating from inside or outside the script) and/or manage requests from the students that will lead him/her to consider script modifications. If the rationale behind the script has been made explicit and constraints analyzed on the intrinsic/extrinsic axis, this will help the teacher in being aware of the consequences of these modifications and provide a basis for his decisions. Typically, extrinsic constraints can be overruled, while intrinsic constraints should not unless the teacher deliberately changes the pedagogical intentions.

Our view of integrated scripts with a teacher as the conductor leads us to consider intrinsic constraints not to be hard limits but rather strong recommendations. The teacher must have the “final cut”: he/she must be given a “bypass” possibility that will allow him/her to act at run time on the different issues listed here before. This, however, should be made as an explicit decision, as the conclusion of an analysis of the script enactment.

As discussed in section 4, making this type of knowledge explicit can also be a first step towards the construction of script engines that would manipulate this knowledge and take decisions or rather, in our view, help the teacher to take some decisions.

3.3 Flexibility for the student

The flexibility given to students varies according to the nature of the script.

- By definition, micro-scripts allow low flexibility. Micro-scripts are based on a fine grained decomposition of tasks, and constrain interactions by prompting turn taking or imposing an argumentation grammar. However, a student who is prompted "Please, provide a counterargument" still has the possibility to type "I agree with all you said", hence escaping from the constraints. Flexibility is lower in micro-scripts but not null.

- Since macro-scripts are based on larger grained activities, they tend to offer more flexibility. For instance, in the ConceptGrid script the role distribution is left open to students: it is not fixed by the script structure and is not constrained by the interface. Differently, in the ArgueGraph, the role distribution obeys intrinsic constraints and is not flexible for students. Their argumentation, however, is not constrained: students have to converge on one answer (intrinsic constraint) but there is no constraint on the way they argue. Similarly, the ConceptGrid brings students to explain concepts because they face somebody who has not read about the concept they need to juxtapose. The interface constraints the number of explanations to be constructed, but not the way explanations are constructed.
- Self-scripts, which are particular cases of macro-scripts, are highly flexible. We refer to self scripts as scripts that, like the RSC script, enable students to define part of their own organization. For instance, in the RSC script, the script fixes the structural phases but allows students to establish the agenda of each phase, decompose tasks into subtasks, select the tools they want to use, etc. This requires making how the script may be negotiated and modified over time explicit, which is described in some kind of meta-script.

4 Implementing flexibility

4.1 General principles

We discriminated intrinsic and extrinsic constraints. Script implementation should allow latitude (for students and teachers) within the extrinsic constraints whilst respecting intrinsic constraints. The implications of this principle to CSCL environments that operationalize macro-scripts are twofold. In terms of design, script designers and teachers should explicitly dissociate intrinsic and extrinsic decisions and document them. In terms of implementation, the ideal platform should respect intrinsic constraints and address extrinsic constraints as flexible issues. The platform should be able to manage the flexibility issues by maintaining the actual interaction patterns in the space of extrinsic constraints around the script, without violating the intrinsic constraints (cf. figure 5). Platforms should thus be addressed as script engines, i.e., active systems that control flexibility, by manipulating the four following components:

- A model of the *script*. This structural description of the script is an abstract but machine-readable description of the script components, for instance using some XML dialect. Most scripts can be defined with a limited number of components (e.g., groups, participants, roles, activities and resources) and mechanisms that capture the dynamics of scripts, i.e., how individual learners are distributed over groups (group formation), how roles, activities or resources are distributed over participants (component distribution) and how both components are distributed over time (sequencing) (Kobbe Weinberger, Dillenbourg, Harrer, Hämälainen & Fischer, submitted). A script engine needs this structural description in order to check if a request for flexibility raises data integrity or other computational problems related to extrinsic decisions.
- A model of the *actual interaction patterns*. This representation captures the emergent organization of the team (e.g., the students' actions or the students' interactions) and their progression in the script (e.g., progress through phases or deliverables). This model must be constructed by analyzing the platform logs (tracing the students' actions) and, eventually, data obtained from students and/or teachers. To facilitate comparisons between the actual interaction patterns and the script, both should be represented with the same scheme (or isomorphic schemes). Capturing the actual interaction patterns is close to the notion student modelling, which has been much developed in research on intelligent tutoring systems (Dillenbourg & Self, 1992), then extended to the notion of group modelling and automatic interaction analysis (Mühlenbrock & Hoppe, 1999; Martinez-Mones et al, 2003). The craft of identifying interaction patterns relies on the

possibility to describe these patterns with interface acts rather than from natural language understanding.

- The script *intrinsic constraints*. These constraints can be considered as the script design rationale, i.e., the principles from which the script has been generated and that must be respected in order to maintain the specific mechanisms that underlie the script. The script intrinsic constraints must be defined by the script designer.
- The script *extrinsic constraints*. These constraints can be considered as contingent decisions, i.e., decisions that could have turned in another way. These constraints are generated by the script designer (parameters fixed in an arbitrary way, etc.) and/or the script implementer (constraints due to the technical features of the operationalization platform, etc.).

The script engine should manage flexibility by permanently comparing the difference between the script and the actual interaction patterns, checking if these differences violate the intrinsic constraints and deciding the next allowed actions (cf. figure 5). The output of the engine is to authorize or not a student's or teacher's action:

- If the requested action is inside the authorized flexibility and is accepted, the script engine must update the different models of the script in order to keep them coherent: for instance, if it allows a team of 4 to continue with 3 members, it will decide how to cope with the missing inputs; conversely, if it accepts an extra member, it needs to ask for his/her inputs. Script modifications need to be reflected in the script visual representation (see below).
- If the requested action is outside the authorized flexibility and is not accepted, the script engine must inform the concerned students and/or teacher.

As suggested here above, managing flexibility may be partly done by the students themselves and of course by the teacher. This requires the script engine to manage a *visualization of the script* (Berger et al, 2001), i.e., a description of what the students have done and what they are expected to do, informing actors about where groups are in the script sequence, what has been done and not, etc. This representation must be updated in real time by the script engine.

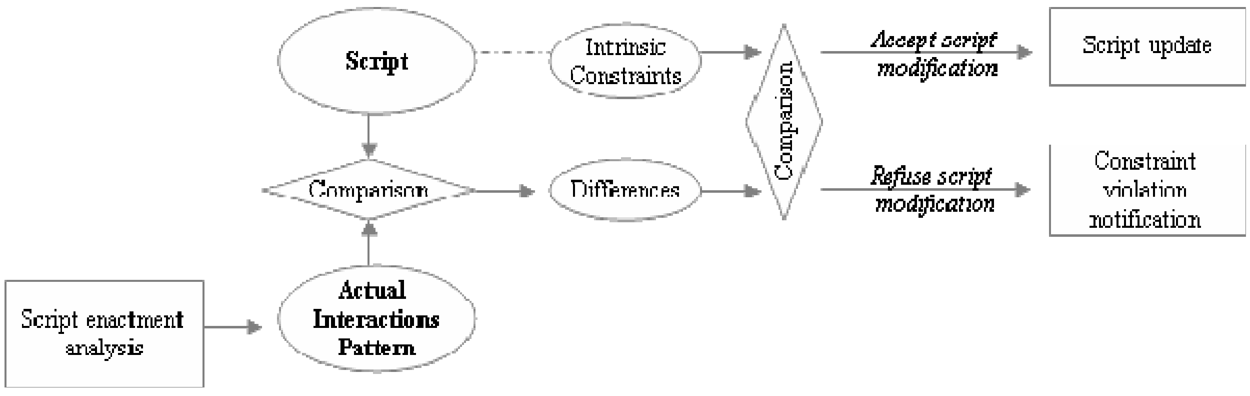

Script
Intrinsic Constraints
Comparison
Accept script modification
Script update
Comparison
Differences
Refuse script modification
Constraint violation notification
Script enactment analysis
Actual Interactions Pattern

Figure 5. Conceptually, managing flexibility can be defined by analyzing if the differences between the script and the actual interaction patterns violate the intrinsic constraints

4.2 Practical solutions: examples

The previous section provided general guidelines to manage flexibility within CSCL scripts, but actual solutions raise more practical issues. Some computational constraints have been described as extrinsic because not belonging to the design rationale, and hence should be bypassed. They can however constitute serious problems.

Let us consider the ConceptGrid script. If a teacher instantiates the script with 4 roles, the script will run smoothly with teams of 4 students. What happens if, when forming groups or while running the script, a team is reduced to 3 members or needs to include a 5th member? The precise group size being not an intrinsic constraint, groups of 3 or 5 should be tolerated. But how will these teams achieve the task without an increase/decrease of individual workload and without technical problems?

Such issues can be addressed by capitalizing on knowledge from experiences and identifying from local solutions what concepts can be conceptualized and re-used (and within what scope). For instance, considering the constraints and flexibility issues that were raised in the context of the ConceptGrid, we have identified different concepts that may be relevant for a variety of scripts:

- Simulated student. For well defined tasks, the role of the missing student can be played by a more or less elaborated artificial agent using mechanisms ranging from retrieving pre-defined solutions to cognitive plausible simulations.
- Duplicated student (or 'clones'). If the student playing role-X leaves the team, the remaining team members are provided with the outputs produced by the student playing role-X in another team or by any student playing role-X in any other team. The latter solution is used in a recent implementation of the ConceptGrid: the students in the incomplete group are presented with the missing concept definitions produced by all other teams and they select the most pertinent one.
- Multi-roles student (or 'role login'). The teacher may ask a student to play two roles if he/she estimates that the workload increase can be managed (for instance because of an easy task, an excellent student or a longer delay). This can be addressed by associating the login procedure with role selection (user-U, role-X; user-U, role-Y).
- Duplicated role (or 'Twins'). For practical (more students than roles) or pedagogical reasons (role that is especially difficult or time-consuming, weak students, etc.), the teacher may ask two (or more) students to share the same role. In a distance setting, this can be done by cloning Role-X as Role-X1 and Role-X2: (user-U1, role-X1; user-U2, role-X2). Role-X1 and Role-X2 have a shared access to Role-X resources and functionalities.
- Multi-students role (or 'multiple login'). If the previous situation occurs in a classroom setting, when students sit together in front of the computer, traceability of individual actions can be maintained if a multiple login procedure (user-U1, user-U2, role-X) enables both student to inform the system there are jointly acting as role-X.
- Opportunistic roles (or 'jokers'). If there are more students than roles, the teacher may leave the group sharing workload with the extraneous member. For instance, within the ConceptGrid script, a neutral role can be created: the student can offload the work (readings, definitions, etc.) of any team member.

Such practical solutions can be seen as potential patches. Teachers should be able to analyze the actual interaction pattern and consider what flexibility means can be used to deal with unpredicted situations that appear to be pedagogically negative. This process should be informed and regulated by the intrinsic/extrinsic concern. As one can see from the examples here-above, these means are dependent upon the platform characteristics. As a matter of fact, it can be noticed that an easy way to describe them is, as we did, to use computing metaphors (relation login/role, etc.). Progress must be made towards the fact that (1) CSCL scripts can be described using pedagogical notions (e.g., groups or roles) and (2) this description is correlated to the computer-science notions mechanisms that ensure their semantics and underlying processes.

5 Conclusions

Scripting collaborative learning is a subtle art where the need to scaffold the emergence of rich interactions should not spoil the natural group dynamics. Computerized scripts, as workflows, relieve the teacher from back office tasks (e.g., managing deadlines, collecting assignments or distributing feedback) but may break the subtle equilibrium between freedom and constraints by increasing the script rigidity beyond pedagogical usefulness. Therefore, we conducted this conceptual analysis of the different types of flexibility required by the actors (teachers and students) of a CSCL script.

The result of this work is the conclusion that the management of CSCL scripts requires manipulating multiple representations of the script: the script to be executed; the actual interaction patterns or emergent organization of teams; the set of intrinsic and extrinsic constraints that result respectively from the pedagogical design and from other decisions and the visual representation of the script for students and teachers. This conclusion already helps in dealing with practical concerns related to the management of CSCL scripts by teachers, whatever the used operationalization platform is.

Based on this conclusion, we have proposed generic guidelines that advanced operationalization platforms should comply with and suggested a general architecture. This approach opens interesting lines of research related to (1) the elaboration of modeling languages to denote the script characteristics we have highlighted and (2) the computer-science issues related to the elaboration of such advanced platforms and, in particular, the diagnosis and then the taking into account of intrinsic constraint violations.

Some of the illustrated arguments for flexibility in CSCL scripts are very down-to-earth. They result from day-to-day practical events illustrated in section 4.2. These practical constraints are inherent to the notion of integrative scripts, i.e., scripts that are conducted in real time by a teacher who thereby articulates several class activities. Such scripts do not occur in a virtual world but in real classrooms and, therefore, the classroom-constraints and events must be taken seriously and reflected in our understanding of CSCL environments.

6 Acknowledgments

This work is partly funded by the Swiss Center for Innovation in Learning and Kaleidoscope, a European Network of Excellence. Thanks to COSSICLE, a Kaleidoscope team working on CSCL scripts, as well as to P. Jermann, S. Parvez, F. Hong, R. Reichert, and R. Hämäläinen.

7 References

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

Berger A., Moretti R., Chastonay P., Dillenbourg P., Bchir A., Baddoura R., Bengondo C., Scherly D., Ndumbe P., Farah P. & Kayser B. (2001) Teaching community health by exploiting international socio-cultural and economical differences. In P.Dillenbourg, A. Eurelings & K. Hakkarainen. *Proceedings of the first European Conference on Computer Supported Collaborative Learning*, pp. 97-105, Maastricht, March 2001.

Betbeder M.L, Tchounikine P. (2003). Symba: a Framework to Support Collective Activities in an Educational Context. In *Proceedings of the International Conference on Computers in Education*, pp 188-196, Hong-Kong.

Delium C. (2003) OSCAR: a framework for structuring mediated communication by speech acts. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies* (ICALT'2003), p. 229-233, Athens (Greece),.

Dillenbourg P. & Jermann P. (to appear). SWISH: A model for designing CSCL scripts. In F. Fischer, H, Mandl, J. Haake & I. Kollar (Eds) *Scripting Computer-Supported Collaborative Learning - Cognitive, Computational, and Educational Perspectives*. Computer-Supported Collaborative Learning Series, New York: Springer

Dillenbourg P. & Self J.A. (1992) A framework for learner modelling. *Interactive Learning Environments* 2(2), 111-137.

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

Hämäläinen R., Häkkinen P., Järvelä S. & Manninen T. (2005). Computer-Supported Collaboration in a Scripted 3-D Game Environment. In T. Koschmann, D.D. Suthers & T.-W. Chan (Eds.)

Proceedings of the 7th Computer-Supported Collaborative Learning, pp. 504-508. Mahwah, NJ: Lawrence Erlbaum.

Jermann P. & Dillenbourg P. (1999) An analysis of learner arguments in a collective learning environment. In C. Hoadley et J. Roschelle (eds), *Proceedings of the Third Computer-Supported Collaborative Learning Conference*, pp. 265-273, Stanford, December 1999.

Kobbe L., Weinberger A., Dillenbourg P., Harrer A., Hämäläinen, Fischer F. (submitted). Specifying collaboration scripts. *International Journal of Computer-Supported Collaborative Learning.*

Kollar I., Fischer F. & Hesse F. W. (in presss). Computer-supported cooperation scripts - a conceptual analysis. *Educational Psychology Review*.

Martínez-Monés A., Dimitriadis Y., Rubia-Avi B., Gómez-Sánchez E., de la Fuente-Redondo P. (2003) Combining qualitative evaluation and social network analysis for the study of classroom social interactions. *Computers and Education.* 41(4), 353-368

Mühlenbrock M. & Hoppe U. (1999) Computer-Supported Interaction Analysis of Group Problem Solving. In: C. Hoadley & J. Roschelle (Eds.), *Proceedings of the Conference on Computer Supported Collaborative Learning*, CSCL-99 , pp. 398-405, Palo Alto, CA, December 1999.

O'Donnell A.M., & Dansereau D.F. (1992). Scripted cooperation in student dyads: A method for analyzing and enhancing academic learning and performance. In R. Hertz-Lazarowitz and N. Miller (Eds.), *Interaction in cooperative groups: The theoretical anatomy of group learning*, pp. 120-141. London: Cambridge University Press.

Palincsar A.S. and Brown A.L. (1984) Reciprocal Teaching of Comprehension-Fostering and Comprehension-Monitoring Activities. *Cognition and Instruction* 1(2), 117-175.

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

Schank R.C., & Abelson R. (1977). *Scripts, Plans, Goals, and Understanding*. Hillsdale, NJ: Erlbaum Assoc.

Schellens T., Van Keer H., & Valcke M. (2005). The impact of role assignment on knowledge construction in asynchronous discussion groups: A multilevel analysis. *Small Group Research* 36, 704-745.

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