Operationalizing macro-scripts in CSCL technological settings

Pierre Tchounikine

LIUM - Université of Le Mans
Avenue Laennec
72 085 Le Mans Cedex 9
France
Pierre.Tchounikine@lium.univ-lemans.fr

Abstract:

This paper presents a conceptual analysis of the technological dimensions related to the operationalization of CSCL macro-scripts. CSCL scripts are activity models that aim at enhancing the probability knowledge generative interactions such as conflict resolution, explanation or mutual regulation occur during the collaboration process. We first recall basics about CSCL scripts and macro-scripts. Then, we propose an analysis of some core issues that must be made explicit and taken into account when operationalizing macro-scripts, such as the reification of some aspects of the script within the technological setting, the strategy within which students are presented with the technological setting and the uncertainties related to scripts and technological setting perception and enactment. We then present SPAIRD, *a model that we propose as a means to conceptualize the relations between scripts and technological settings used to operationalize them. This model dissociates four points of view on the script (structural model, implementation-oriented model, student-oriented models and platform specification) and the underlying design rationale (learning hypothesis, pedagogic principle and design decisions). In order to exemplify* SPAIRD*'s usefulness we propose examples of how it allows drawing general propositions with respect to the couple script + technological setting. Finally, we present an analysis of current state-of-the-art technological approaches with respect to this conceptualization, and research directions for the design and implementation of technological settings that present the properties identified in our analysis. In particular, we study the interest of model-driven approaches, flexible technological settings and model-based script engines.*

Keywords: CSCL macro-scripts, operationalization, technological setting, computer science.

1. Introduction

As defined in Kobbe & al. (2007), *CSCL scripts* are activity models which aim at structuring and supporting collaboration among distant students or co-present students whose action or interaction is (at least partially) mediated by a computer-based system. A CSCL script typically describes the task to be achieved by students and issues, such as how the task is to be deconstructed into subtasks, the sequencing of these subtasks, the role of each student, the constraints to be respected and the computer-based system to be used by the students. From a general point of view, CSCL scripts take their origin in the fact that the effects of collaborative learning depend on the quality of interactions that take place among group members (Dillenbourg, 1999). CSCL scripts aim at enhancing the probability that knowledge-generative interactions, such as conflict resolution, explanation or mutual regulation occur during the collaboration process (Kollar & al., 2006), (Kobbe & al., 2007). As defined in Kobbe & al. (2007) and Dillenbourg & Jermann, (2006), CSCL scripts can be dissociated into CSCL macro-scripts and CSCL micro-scripts. CSCL *macro-scripts* are coarse-grained scripts that follow a pedagogy-oriented approach and emphasize the orchestration of activities. They differ from *micro-scripts*, that are finer-grained scripts following a more psychological and bottom-up approach.

With respect to CSCL scripts, the role of the computer-based system is twofold. First, the computer-based system is supposed to provide the technological means required by the script. For instance, the computer-based system must provide the communication functionalities that will allow students to interact, or the specific modeller that will allow them to achieve the modelling task described by the script. Second, the computer-based system can also participate in structuring and constraining the students' process. For instance, it can be designed to contribute to structuring the sequences of activities or the way students engage in individual and collective activities by introducing a specific dataflow or workflow, or provide communication functionalities that impact students' interaction by imposing sentence-openers or turn-taking structures. Computer-based systems used to operationalize CSCL scripts can be standalone tools (e.g., a communication tool or a shared graphic modeler), all-in-one systems (i.e., systems that provide within a dedicated integrated interface different functionalities, e.g., an interactive shared simulation coupled with a chat) or platforms (i.e., a set of functionalities/tools made accessible through a script-related interface or a generic interface such as the one provided by Learning Management Systems). We will use *technological setting* as a general notion that covers these different types of software.

CSCL scripts raise different research questions, such as defining, modelling and operationalizing scripts (cf. for example the work presented in Kobbe & al. (2007)), experimenting scripts' effects (cf. for example the work presented in Weinberger & al. (2005)) or studying the issues related to their use by practitioners (cf. for example the work presented in Hernández-Leo & al. (2005)). Our work is related to the operationalization of CSCL macro-scripts. We refer to *CSCL macro-script operationalization* as 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.

In this article we focus on the way CSCL macro-scripts' technological settings should be thought of. The general long-term objective of our research is to develop principles, methods and technologies for the design and implementation of such technological settings. Within this perspective, we think that, as a premise, there is a specific issue in conceptualizing the interrelations between macro-scripts and the technological dimensions of their operationalization. We refer to a conceptualization model as a model that highlights basic notions and issues, and provides a kind of pre-structured map for relating pedagogical issues and issues of technology design. This is an intermediary level between technological-independent descriptions of the script and precise modeling languages. Such an intermediary level allows script and technological-setting designers to share an intermediation (or boundary) model to communicate and think with whilst preventing to go in a too-straightforward way into a specific fine-grained modeling language. Such a conceptualization is a premise because building technological settings to support macro-scripts is not just a technological issue, i.e., building a computer-based system that respects a definitive set of specifications that are straightforward implications from the macro-script technologically-independent description; some design decisions are related to both educational and technological issues, with these two dimensions influencing each other. In order to address this issue, it is necessary to provide a general picture of the relations between CSCL macro-scripts and technological settings and how these are thought of, as a conceptual means for tackling these two dimensions in an articulated way.

Within this perspective, we propose the following contributions in this article:

1. An analysis of different issues related to technology that must be taken into account when operationalizing macro-scripts: how technology can be used to reify some features of CSCL macro-scripts; strategies within which students can be presented with the technological setting, and their underlying assumptions; uncertainties related to macro-scripts' perception and enactment (in particular, as related to the dimensions related to technology).
2. A conceptualization model, i.e., a model whose objective is to make salient notions to be taken into consideration when considering CSCL macro-scripts' operationalization. This model, called SPAIRD (for Script-PlAtform Indirect Rational Design), helps in conceptualizing the relations between the script and the technological setting by dissociating four points of view on the script (structural model, implementation-oriented model, student-oriented models and technological setting specification) and making designers make explicit the underlying design rationale (learning hypothesis, pedagogic principle, design decisions). This provides a general understanding of issues to be considered, which is helpful by the fact it makes issues to be put on the designers' worktable explicit, and provides an intermediation model that may facilitate how (non-technical) educators and computer scientists can collaborate to address macro-script operationalization. In order to exemplify SPAIRD's usefulness we propose examples of how it allows drawing general propositions with respect to the couple macro-script + technological-setting.
3. With respect to this conceptualization, an analysis of current state-of-the-art technological approaches, and research directions for the design and implementation of technological settings that present the properties identified in our analysis. In particular, we emphasize the interest of model-driven approaches, and of flexible model-based script-engines.

When designing CSCL settings, the properties of the technological setting are but a dimension. Taking a wider perspective, Kirschner & al. (2004) propose to focus on interaction design and consider technological, social and educational affordances. Strijbos & al. (2004) propose a methodology for interaction design based on six steps and five critical elements (learning objectives, task type, level of pre-structuring, group size and computer-support). Similarly, from an analysis point of view, researches taking their origins in Vygostki's works (e.g., Engeström, 1987) highlight that technological settings should be thought of in terms of mediating tools, and that a wider activity-centered analysis is necessary. Not misunderstanding this, we think the technological and usage dimensions of the computer-based system that students use when enacting the script require specific attention. Technology is not "neutral" in the sense that any given program (e.g., a modeling tool or a communication tool) carries epistemic primitives via the way it presents users with the data or via the objects that users can manipulate within its interfaces. Similarly, the way technological settings integrate different functionalities within an interface or support/constrain students by a specific workflow has an impact on the way students perceive the script and on their enactment of the script (although not necessarily the one that was anticipated), and are thus of importance. As highlighted in Jones & al. (2006), "Seen from the practice of design, technologies do indeed embody features and properties and they also carry meaning. Having been designed with certain purposes in mind, certain understandings of communication, interaction and collaboration were embedded in the design process." Within CSCL research, computer science has thus two roles: on the technological side, to propose technological means to operationalize CSCL settings; on the conceptual side, on the basis of and in interaction with educational and usage research, to elaborate meaningful conceptual frameworks that contribute to the understanding of operationalization processes. This latter dimension is important to allow operationalization processes that take into account dimensions related to the use of technology and the used-technology specificities, to define informed specifications of technological settings, and to inform the analysis of scripts' enactment and the re-engineering of scripts. The work presented in this article is of this conceptual nature, and takes place within this perspective.

This article is organized as follows. In Section 2 we recall some basics of CSCL macro-scripts. In Section 3 we pinpoint and analyze three dimensions that we have identified as core issues to be disentangled, made clear, and taken into account when operationalizing macro-scripts: the reification of macro-script issues by the technological setting, the principles that underlie the way students are presented with the technological setting, and the uncertainties related to macro-script perception and enactment. In section 4 we present SPAIRD, the conceptualization model we propose as a general understanding of the notions to be taken into consideration when considering the technological dimensions of macro-script operationalization. In order to exemplify SPAIRD's usefulness we propose in §4.4 examples of how it allows consideration of design questions involving dimensions related to both the script and the technological setting. Finally, in Section 5 we first analyze different current approaches to macro-script operationalization and how they can be characterized with respect to the issues raised in this article, and then discuss general directions for future CSCL macro-script technological settings as model-driven computational engines.

In this article we focus *CSCL macro-scripts*. In order to simply the text we will drop the "CSCL" and/or the "macro" when not ambiguous.

2. Basics about CSCL scripts

2.1. CSCL scripts

On the basis of the reference article Kobbe & al. (2007) and the works compiled in Fischer & al. (2007), we refer to a *CSCL script* as a model that specifies the specific collaborative activities that a group of students are expected to engage in within a computer-mediated setting, and the associated supports and constraints. As discussed in Kobbe & al. (2007), CSCL scripts take their origin in the scripted cooperation approach (O'Donnell, 1999). They foster collaborative learning by shaping the way students will engage in interactions such as asking each other questions, explaining and justifying their opinions, articulating their reasoning, or elaborating and reflecting on their knowledge. For this purpose, CSCL scripts describe and orchestrate individual and collective tasks, the way students should distribute roles, the rules to be respected (e.g., deadlines or mandatory means), and the computer-based technological setting. Within this context, computers are both a support for students to achieve their tasks, and a means to coordinate students' activities in a way that is coherent with the script principles. CSCL scripts are a key mechanism by which computers may support collaborative learning (Jermann & Dillenbourg, 1999; Kollar & al., 2006; Fischer & al., 2007).

In this article we consider CSCL macro-scripts as a kind of pedagogical method to be used in open settings (schools, universities; Dillenbourg & Jermann, 2006). CSCL scripts can vary from rather psychology-oriented scripts (micro-scripts) to rather pedagogy-oriented larger-grained scripts (macro-scripts; Kobbe & al., 2007). A micro-script models a process to be internalized by students, and is designed to scaffold the interaction process *per se*. As examples, micro-scripts will make a student state a hypothesis and will prompt a peer to produce counter-evidence, or will constrain interactions by prompting turn taking or imposing an argumentation grammar (Kollar & al., 2006). A macro-script is rather a pedagogical method that aims at producing desired interactions. Macro-scripts are based on indirect constraints generated by the definition of the sequence of activities, the characteristics of the groups or the technological-setting proposed functionalities and/or interface. Macro-scripts aim at triggering high-order thinking activities involving complex cognitive processes such as elaborating on content, explaining ideas and concepts, asking thought-provoking questions, constructing arguments, resolving conceptual discrepancies or cognitive modeling (Kobbe & al., 2007). The macro/micro script differentiation is further discussed in §2.2, after examples have been given.

2.2. Examples of CSCL macro-scripts

We present here below two examples of macro-scripts. Other examples can be found in Kobbe & al., (2007) DiGiano & al. (2002) or Fischer & al., (2007).

The Concept-Grid script (Dillenbourg, 2002) is a subclass of the Jigsaw family of scripts, i.e., scripts that are based on making individual students manage some partial knowledge and then prompting them to collectively solve a problem that necessitates knowledge from each of them. Concept-Grid includes four phases. (1) Groups of four students have to distribute four roles among themselves. Roles correspond to theoretical approaches of the domain under study (e.g., learning theories). In order to learn how to play their roles, students have to read a few 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 define the relationship between grid neighbors. The key task is to write 5 lines that relate or discriminate between 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. (4) During the debriefing session, the teacher compares the grids produced by different groups and asks them to justify divergences. The core functionality of the computer-based system that supports Concept-Grid operationalization is the grid-editor that provides both support (what students must do is made clear by the line/column structure; specific editors are provided) and constraints that impact students' activity (the number of relations to be defined is not open but constrained by the line/column structure and the ratio number of definitions/number of cells; the limited length of the text to be edited constrains students to synthesize their analysis; Dillenbourg, 2002; Hong & Dillenbourg, 2007). The latter version of the system supports all aspects of the script edition and enactment (role distribution, access to documents, etc.), including functionalities that help the teacher in tuning the script and regulating the process (Hong & Dillenbourg, 2007).

The Crossing-Analyses script aims at triggering interactions among pairs (elaborating on content, explaining ideas and concepts, asking thought-provoking questions, constructing arguments, resolving conceptual discrepancies) by asking groups G_i to elaborate an analysis A_i, reorganize groups differently, and then ask a group G_j to elaborate on A_i (and vice versa). This general principle can be used to create different scripts: groups in the first phase can be limited to one student when groups in the second phase are composed of several students, the objective being to make the group elaborate on the basis of its individuals' productions; groups in the second phase can be composed by mixing students from the first phase groups, with the objective of making individuals explain the collective productions of their origin group; etc. The RSC script (Betbeder & Tchounikine, 2003) is an example of a large-grained instance of the Crossing-Analyses script. RSC is based on three phases (Research - Structure - Confront) which can be repeated several times, the output of a phase being the input of the next: (1) each student has to freely research on the Internet some information on a given topic and become familiar with it, 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 in order to analyze educational Websites; (3) the individuals are grouped and have to elaborate a collective construction from the individual productions, e.g., confront the individual grids and collectively construct an analysis of some Websites. The computer-based system that supports RSC operationalization provides students with different forms of support: access to the different phase's descriptions; means to discuss and edit a plan of how they intend to tackle each phase's different subtasks (shared plan editor and task editor coupled with a synchronous communication tool); awareness functionalities such as means to declare their individual advancement; etc. It also carries constraints. For instance, accessing the interface dedicated to realizing a task is conditional on the fact that the corresponding task has been collectively described previously, which puts pressure on the students to organize themselves explicitly.

As one can see from these examples, macro-scripts can address fully or partially mediated situations. RSC is designed for distance learning students and completely meditated by the proposed platform. Concept-Grid embeds phases that take place face-to-face and can be partially or completely mediated (e.g., face-to-face discussions can be replaced by on-line discussions). In the rest of this article we will focus on the issues related to the operationalization of scripts through technological settings, not misunderstanding however that some of them can be addressed through mixed modes.

The macro-script/micro-script differentiation denotes both levels-of-granularity and matters-of-concern issues. Dillenbourg & Tchounikine (2007) exemplify the distinction with scripts that aim at raising argumentative dialogues. Typically, considering argumentative dialogues, works referring to the macro-script notion aim at setting up conditions in which argumentation should occur (e.g., bring students to build shared answers as in ConceptGrid or pair students with opposite opinions as in the ArgueGraph script (Jermann & Dillenbourg, 2003)) while works referring to the micro-script notion aim at scaffolding the interaction process *per se* (e.g., when a learner brings an argument, the script prompts his or her peer to state a counter-argument (Kollar & al., 2006)). These two examples first differ by their granularity: a phase in a macro-script is an activity that may last for several hours (or several weeks for a script such as RSC) while when in a micro-script, it may be a single conversational turn. This degree of granularity is not binary, and the micro/macro distinction can be considered in this dimension as a continuum. However, these two examples have very different statuses: ConceptGrid and ArgueGraph are pedagogical methods and, when designing and tuning the script and its technological framework, the emphasis is on how to make these designed issues to be adopted by the students. Differently, Kollar et al.'s script and work emphasis is on if and how the model of dialogue conveyed by the script is internalized by students. This is a different perspective, and is related to different methodologies. Here again this distinction is not binary. For example, a script such as RSC is “macro” in terms of granularity but some of its features are nonetheless expected to be internalized (e.g., students are expected to internalize concepts such as “plan” or “tasks” and/or to build an explicit organization). It can thus be considered that there is a continuum in this dimension also, but the emphasis and matters-of-concerns are different. In this article we focus on macro-scripts as the type of scripts where our concern (interrelation script/technological framework) are the core issue. From this perspective, works on macro-scripts can be put into relation (as being from similar level/matters-of-concern, although of slightly different objectives) with works related to identifying and/or using for design *collaboration patterns* as defined in Wasson & Morch (2000), i.e., recurrent sequences of interaction among members of a team that satisfy established criteria for collaborative behaviour (Wasson & Morch, 2000; DiGiano & al, 2002). As micro-scripts of course also require taking into account some operationalization technological dimensions, some aspects of this work may also be of interest with respect to micro-scripts.

2.3. From CSCL scripts to technological settings

At a general level, CSCL scripts can be described and understood independently from technological issues. As an example, Kobbe & al. (2007) propose a model that allows describing scripts in terms of structures (resources, participants, groups, roles, activities) and mechanisms (task distribution, group formation and sequencing). Using this model, the authors propose an abstract description of different scripts reported in the literature, descriptions that can be reused and/or refined and adapted according to a given context.

Within their technological dimensions, macro-scripts are based on the use by students of computer-based systems providing functionalities such as mediated-communication functionalities (e.g., possibilities for synchronous communication, asynchronous communication, file-exchange or awareness) and task-specific functionalities (i.e., functionalities dedicated to the particular tasks to be achieved, e.g., a simulation or an editor of models). From a technological point of view, this can correspond to different types of computer-based systems, such as all-in-one systems (i.e., systems providing within a dedicated integrated interface the different required functionalities), platforms (i.e., systems providing access, through a common interface, to the required tools or web services defined as building components), or a set of separate stand-alone tools (e.g., a chat tool). Integrative software such as all-in-one systems and platforms can propose dataflow and/or workflow functionalities, i.e., structure the way students can access data and/or functionalities. Section 5.1. presents an overview of current major approaches.

The operationalization of a macro-script, i.e., going from an abstract description of a script to an effective setting, can be addressed in very different contexts/manners. As in this research we address a general conceptualization level, we will consider the following canonical situation. Given a set of pedagogical objectives and the considered pedagogical context, it is decided to provide students with a collaborative task and to structure the way students will tackle the task by using a script that makes explicit a sequence of phases, the input and output of the phases, the roles of the students, and some constraints. The used script can be an original construction or an instance or variation of an abstract script reported in the literature. When an abstract script is used, a prototypical process can be to edit the script (i.e., modify the abstract script to better fit the present pedagogical goals, e.g., change the order of phases or add/remove a phase), instantiate the script (i.e., “fill” the abstract script with the relevant content), and finally set up the session (i.e., specify features such as the group composition or the group composition procedure, or the duration for each phase; Dillenbourg & Tchounikine, 2007).

This canonical situation is subject to many variations. For instance, the fact the process is managed by a teacher rather then a multidisciplinary team introduces issues related to the teacher’s competence and ability in managing different levels of abstractions and/or the technological dimensions. A teacher may also address his pedagogical objectives by different means more or less intertwined with the script, which can be but a part of, or overlap with, other social protocols. It can also be noted that structuring the way students will tackle the proposed task can correspond to different situations: (1) structure as a support (i.e., as a means to succeed in a complex task that would not be successful without the script); (2) structure as a constraint (i.e., as a means to force students to a given behaviour). These two dimensions are not exclusive one from the other (structure as constraint being one implicit way of providing structural support), and may correspond to different realities for teachers and students (for instance, students can have no need of the proposed support and develop their own approach, what was meant as a support becoming a constraint; in such a case this constraint can however still be of a positive effect with respect to the overall pedagogic objective, but can also become only counter-productive). As our objective is to elaborate a general conceptualization model, we consider the aforementioned canonical situation, however not misunderstanding this variety and its implications.

Considering macro-scripts, specific attention is required to the fact that the design of macro-scripts and their associated technological settings follows a razor’s edge. The purpose of a script is to introduce structure and constraints that will shape collaborative interactions. As emphasized in Dillenbourg (2002), if this 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. Macro-scripts carry the risk of over-scripting collaboration, i.e., constraining collaboration in a way that makes it sterile (Dillenbourg, 2002). This issue must be kept in mind when considering the questions raised by the operationalization of macro-scripts, such as: How can one use both the script and the technological setting to make students perceive and enact the script according to the pedagogical objective? How should the technological setting reify or take into consideration the way the script sequences different phases? What features of the technological setting (as related to the script) should be modifiable if the actual interaction differs from expectations, or if some unpredictable events arise? What flexibility students should be provided with in order not to be over-constrained whilst keeping the script's *raison d'être* and remaining coherent with the pedagogical objectives? Such design questions, whose answers will impact the script enactment, are related to both educational and technological issues, these two dimensions influencing each other.

Our work aims at contributing to making these issues clearer, as a way to facilitate how (non-technical) educators and computer scientists can collaborate to address them. Macro-scripts are used in different social and pedagogical contexts, and there is no point in attempting to define a canonical operationalization process and associated guidelines. Our objective is, rather, to propose a conceptualization that provides a general understanding of the different notions that can/should be considered when addressing the operationalization of a script, as a cornerstone for additional and more precise/instantiated specific studies.

3. Implementing macro-scripts

In order to elaborate a general understanding of the different notions that should be considered when addressing the technological dimensions of the operationalization of a script, a first step is to disentangle different design concerns, and not only technologies. At this level, and as a premise, we think the role that designers attribute to technology and their view on the use of technology should be made explicit, and not kept implicit within the head of designers.

Technologically-related design decisions consider issues such as what functionalities would be useful or should be used by students, if and how these functionalities should be integrated and/or articulated within a common interface or, when it is considered that no pertinent technology already exists, what the specifications of the software to be built are. When considering these design issues, the problem to be solved can however be thought of in different ways. From this perspective, it is particularly important to dissociate two general points of view: (1) the considered problem is that of providing students with the functionalities that are necessary to achieve the tasks proposed by the script; (2) the considered problem is to continue the objective of structuring students' collaboration by offering technologies whose properties have been studied according to the script and the targeted support and constraints. These two points of view are not contradictory, the latter addressing a problem that includes the one addressed by the former. However, they denote different concerns, and lead to the taking into account of different issues. In particular, they heavily impact to what extent technological settings are supposed to reify some aspects of the supports and constraints targeted by the script, and the strategies within which students are presented with these technological settings.

Moreover, seen from the perspective of usage, macro-scripts create socio-technical settings. Technology impacts the script enactment, but this impact is however not necessarily the one that is expected, in particular because of the uncertainties of how students will perceive and use the technological setting. It is thus important to take into consideration not only the script and the technological setting as considered by designers, but also the phenomena related to the effective use of technology.

In this section we disentangle and make explicit prototypical approaches to how technology can be used to reify some script issues (§3.1) and how students can be presented with the technological setting (§3.2). Our claim is not that all works fall in one or another of the prototypical approaches we highlight. Rather, the objective is to propose prototypes as a way for designers to make explicit their way of thinking with respect to these issues (by reference or opposition to one or another view, or blending views). We then list different issues (related to technology) that may contribute to create uncertainties related to macro-script perception and enactment (§3.3), and finally propose a discussion (§3.4). In this section we remain at the level of how the link script/technological settings can be thought of and addressed. A more focused analysis of how different levels of modelling can allow addressing different support functions is discussed in Section 5.1, after we have presented our SPAIRD conceptualization model.

3.1. Reification of script issues within technological settings

Focusing on (1) providing students with the functionalities that are necessary to achieve the tasks proposed by the script or on (2) continuing the objective of structuring students' collaboration can lead to differing considerations of the particular properties of the functionalities or tools provided to students, and of how they are integrated.

Detailed properties of the used software

Let us consider for instance the fact that a script requires students to engage in argument synchronously. Such a requirement can be thought of as the need to make some synchronous textual communication functionalities available for students. The level of support and constraint that is addressed is: "allow synchronous exchanges of messages," which indeed allows exchanging arguments. This can be implemented by providing a basic chat tool. Differently, the operationalization process can be thought of as the need for encouraging students to make their arguments explicit, or to relate their messages to the task at-hand. This would require not just any communication tool, but to consider what is the specific support proposed by structured communication tools, such as Belvedere (Suthers & Weiner, 1995), Oscar

(Delium, 2003), Comet (Soller, 2001) or C-Chene (Baker & al., 1997), and how this support complies with the script objective and the overall script operationalization. Within this approach to operationalization, the properties of the communication tools are considered as means to a specific impact, correlated with the script objectives. As highlighted before, this impact can be considered as, or appear to be, a support (tools help students to formulate arguments) or a constraint (tools force students to structure there messages as arguments), and the effective uses and impacts must be specifically studied. Our point is not to advocate the uses or advantages of structured/unstructured communication tools, but to illustrate the fact that a given feature of a script can be addressed with different matters of concern, and that these matters of concern impact how the detailed properties of the used software will be considered and taken into account or not.

As another example, in the Concept-Grid script (cf. §2.2), students are presented with a 4x4 table to fill that reifies *de facto* the script's basic principle: pairs are presented with a line/column shared editor that suggests a common text is to be edited, and that this text must match the notions denoted by the corresponding line and column. This grid-editor tool is a key element of the script operationalization: it forces students to analyze and relate juxtaposed concepts, imposes a large number of connections by fixing the ratio between the number of cells and the number of concepts to be entered in the grid, and limits the length of explanations. These different constraints have an impact on the students' interaction (Hong & Dillenbourg, 2007). The technological choice continues the script overall objective by reifying part of the script principles.

Integration of functionalities or tools

Macro-scripts are based on sequencing different phases associated with different tasks or subtasks. Therefore, they generally require presenting students with different functionalities/tools. Here again, the integration and/or articulation of these different functionalities/tools can be thought of in different ways. Integration can relate to two dimensions: accessibility (i.e., the way functionalities or tools are made accessible to students) and interoperation (i.e., the way functionalities or tools can be bound together in order to propose integrated service, e.g., implement a data-flow that makes some data produced in a given phase and/or by an individual made accessible as input for another phase and/or another individual). This is related to the way the process dimensions of the script are taken into account.

Let us consider for instance a script stating that students should be presented with means to access some pedagogical resources, share some intermediate results with peers, communicate with peers, collaboratively build a text and deliver the final result to the teacher.

The technological dimensions of such a script can be addressed by presenting students with an open access to a pool of separate standalone tools providing the functionalities required by the script, in this case a file-exchange tool, a chat or a forum, and a collaborative whiteboard. Similarly, another approach is to present students with an all-in-one system or a platform that provides through its interface a common entry-point to the different functionalities or tools. This is integration in the sense of facilitating access to functionalities or tools, as natively proposed by Learning Management Systems, for example. In both cases, the role assigned to the technological setting is limited to that of providing the means that are necessary for the realization of the tasks defined by the script. This can be seen as projecting the script on the technological plan (projection in the mathematical sense, i.e., reducing the number of dimensions of a structure). What is considered at the technological level is an implication of the script in terms of what functionalities/tools should be made available. However, the process dimensions of the script, such as the sequencing of activities, the link between the output of a task and the input of some other, or the grouping issues are not captured.

Alternatively, such a requirement can be addressed by presenting students with an interface that articulates the access to tools/functionalities according to the considered script. This is integration in the sense of correlating the process features of the script and the technological setting. This can be done at different levels of granularity. As an example, the platform used to operationalize the RSC script (Betbeder & Tchounikine, 2003) provides access to the different tools to be used by students. These tools are however not made accessible all at once via a general menu, but according to the script sequencing and its objective. For instance, students are guided and constrained by the fact they can only access the functionalities to be used to achieve a task after they have defined how they intend to tackle this task and have deconstructed it into subtasks and delegated these to specific individuals or subgroups, or by the fact the platform manages the data-flow between the different phases. The platform also provides integrative interfaces suggesting targeted behaviors (e.g., coupling a shared modeler and a chat within the same screen in order to incite students to build a model collectively). As another example, Haake & Pfister (2007) propose a workflow-like approach within which the script is interpreted and run by a software engine that prompts students according to the script sequencing, which allows the system to control access to data/functionalities. These two examples illustrate different extents and different implementation approaches to assigning to the platform the roles of integrating functionalities or tools according to the script principles. Within this view, the platform is assigned the roles of providing the technological means *and* influencing the students' process and behaviour. The underlying idea is that platforms should not only allow the script enactment but also guide this enactment by reifying part of the process suggested by the script. The students are presented with an interface that is not generic as in an LMS, for example, but script-related.

The support functions that can be proposed by the platform that address both the students (e.g., guiding, scaffolding or providing awareness functionalities) and the teachers (e.g., graphical tools to model the script, support to check the structure of a model, ability to simulate a model or automated generation or tuning of the platform from the script model) are directly related to the informedness of the script and platform models (Miao & al., 2005). This is further discussed in Section 5.1.

3.2. Strategy within which students are presented with the technological setting

Focusing on (1) providing students with the functionalities necessary to achieve the tasks proposed by the script or on (2) continuing the objective of structuring students' collaboration can also lead to consider differently the strategies within which students are presented with the technological setting.

The operationalization of the script can be thought of as offering students technological means within self-service conditions. Such an approach is coherent with addressing the problem of providing students with the functionalities required by the script. Within this view, the script can provide guidance or hints on how to use these functionalities or corresponding tools, but there is no technological decision related to the objective of constraining students in their use of the provided technology. This can for instance be addressed by platforms such as LMS. In settings where students are technologically autonomous, it can even be considered that they can find themselves the appropriate means.

Alternatively, the operationalization of the script can be thought of as making students use the technology that designers/teachers want them to use. Such an approach is coherent with the fact that this technology is considered as providing support and/or constraints in line with the objectives of the script.

When the objective is that students should use a given technology, a key question to be answered at design time is: what are the reasons that will make students to use this technology? Different options exist: because they are asked to do so (it is part of the didactical contract (Brousseau, 1998)) and this is considered as a sufficient reason; because they have no other means; because it appears, or it is possible to convince them that, it is more useful to achieve the task they are proposed with; because they have no reasons to use another technology; etc. Design decisions (related to both how the script is tuned and how the technological setting is defined and presented) should consider these reasons with respect to the setting. Different issues must be taken into account, such as the extent to which students are used to using technology and their level of technological autonomy (which impacts to what extent they are inclined to use a given technology or how pro-active they are in deciding what technology they want to use) or the fact that the process is monitored by the teacher and the level of granularity of this monitoring (which impacts the way teachers can be pro-active in controlling what technology is used and how). As said before, when considering these design decisions it must be kept in mind that the extent to which the use of the technology is a pedagogic requirement must be put into balance with the fact that forcing students to use a given technology can become pedagogically counterproductive.

The Concept-Grid and RSC scripts (cf. §2.2) can be used to highlight how settings can be different one from another. In Concept-Grid, students are presented with a task that can only be achieved with the provided technology: they must fill the grid with the Concept-Grid editor. This is an example where the provided technology is the only way to match the script requirements (in this case because the task is explicitly linked to a particular tool). In such a case, not using the provided technology is not an option, independent of the fact that the process is monitored step-by-step by the teacher or that students would prefer to use other means. In RSC, students (University level) are presented with different means to interact, organize themselves, share knowledge and elaborate the collective output. However, the script involves distant students, and goes on for several weeks. Students are asked to use the technology that has been designed to support them and, in general, do so. Interviews however revealed that this was to a certain extent linked to the fact they wanted (or acknowledged that they were supposed) to "play the game," and used this technology as part of the didactical contract. Some groups organize themselves in a way which is coherent with the proposed technology and, as using the technology is not a problem and is a demand from the teachers, they do so. Some other groups, however, organize themselves in a way that makes the proposed technology become a constraint rather then a support. In such cases they generally become pro-active and contextually adopt the means that are the more useful to them.

In a coarse-grained script run in an open setting and with autonomous students (e.g., RSC), the operationalization of the script must thus be thought of as providing suggestions, i.e., attempting to create conditions which favor the fact that students will use the targeted means. It is necessary to acknowledge the uncertainties related to the achievement of this objective, and the fact that students may use different means than the ones that are provided, or may use these in different ways. Technology can be used to introduce constraints, e.g., linking input/output of phases or constraining access to some data or functionalities/tools. The relevance of these constraints (as with all other constraints) is to be studied carefully. For instance, when linking a task and a tool as the only means to match the script requirement, what using this tool implies in terms of behaviour should be examined. As an example, in the Concept-Grid script, what is technically imposed is the fact the students' answers are entered in the grid and respect its constraints. This constrains but does not say anything about the students' effective process and interactions while filling the grid.

From a technological point of view, presenting students with the functionalities/tools they are supposed to use given the script sequencing can be addressed by hand (i.e., orchestrated by the teacher) and/or by the way functionalities and tools are integrated (accessibility dimension, cf. §3.1).

3.3. Uncertainties related to perception and enactment

Associating a macro-script with a technological setting is a particular case of human activity instrumentation. As such, it is subject to different phenomena related to instrumentation in general and to macro-scripts specificities in particular. Here we highlight three issues related to (1) the perception and use of technology, (2) the fact that one might have to deal with unpredicted events and (3) the fact that students may develop self-organization. These issues may apply to different extents, and may be interrelated.

Perception and use of technology

A general difficulty of designing technological means to support students involved in macro-scripts is that technological-setting designers have limited control on how their designs will be enacted.

Following the ergonomic distinction between the notions of *task* (the prescribed work) and *activity* (what people actually do), Goodyear (2001) emphasizes the fact that teachers set tasks and students interpret the specifications of the task, their subsequent activity being a more or less rational response to the task. Activity is related to the task but also to other dimensions (e.g., students' effective motivations or perception that is developed by the students of the task to be achieved and the provided technological setting) that evolve in time, and are interrelated within systemic relations. The activity that will emerge from the confrontation of the students with the task and the technological setting is subject to different contingences that may render it unpredictable in its details.

The unpredictability of usage is partially explained by the concept of affordance. The importance of this concept has recently been raised in the context of CSCL (Jones & al., 2006). First introduced by Gibson and then popularized in the Human-Computer Interaction (HCI) community by Norman (Norman, 1999) with a slightly different definition, the affordance notion denotes the natural or design aspect of an object which suggests how the object should be used (see McGrenere & Ho (2000) for a comprehensive compared analysis of the affordance notion different definitions, and Jones & al. (2006) for a CSCL point of view). The affordance notion helps in understanding that the fact that designers have limited control over how their designs will be enacted is not a matter of "good" or "bad" design. The characteristics of the technological setting will be picked up in different ways by students, who will appropriate them according to their purposes, and in context.

More generally, considering that a computer-based system (a platform, a tool) is appropriate for students on the premise that it has been designed with respect to the task to be achieved by these students is a rather techno-centred view. Applied to the context of students confronted with socio-technical settings as a particular case of instrumented activity, theoretical frameworks such as the activity theory (Engeström, 1987) or the instrumental-genesis theory (Rabardel, 2003) help in understanding that students take advantage of the means that seem best adapted to them in the context of their activity. Within this perspective, designers create *artifacts* on the basis of how they imagine their future use. An artifact only becomes an *instrument* for its user, in the context of his activity, by the fact it allows this user to achieve the tasks he considers, and in the way he considers them: it is the user that gives the status of instrument to the artifact. According to Rabardel, the instrument can thus be seen as constructed from the artifact (the technical object), but also from the user that assigns it some functions in the context of its activity, in a double process of *instrumentation* (adaptation of the user to the artifact constraints) and *instrumentalization* (attribution by the user of functions to the artifact, functions that may correspond or differ from those anticipated by the designer). An instrument must thus be considered as composed of a technical dimension (the artifact), originating from the design process, and a psychological dimension (the usage schemes, specific to the user and/or socially defined) developed by the user, in use. Works in ergonomics on this artifact/instrument dichotomy demonstrate that the artifact impacts but does not define the instrument (Rabardel, 2003). In other words, software functionalities and properties must not be considered as passively received by actors in a form that corresponds to those that underline their design, but as co-constructed by these actors, in the context of their activity, according to their expectations and needs, and thus with psychological, historical and cultural dimensions. From a software design point of view, this suggests the interest of end-user tailorable software (cf. §5.3).

These works help in understanding that students' perception and enactment of CSCL scripts and their use of the provided technological means are intrinsically situated, and therefore difficult to predict. From a technical point of view, a computer-based system is associated with functionalities (a chat allows synchronous exchange of text-based sentences between different computers; a shared modeller proposes a set of notions that can be manipulated, e.g., "definition" or "argument", means to organize them graphically on the interface and means for different connected users to access the model; etc.). These functionalities can be range from general presentations (as described previously) to formal and unambiguous representations. This functional dimension however only defines a technological offer. The fact that students appropriate to themselves the underlying assumptions (e.g., that students will use the presented shared modeller, will associate the modeller notions with a semantics that is similar to that of the designers' or will interact while editing the model) is not a given.

To what extent students are prone to develop unexpected perceptions and/or uses of technology is a question that has no general answer, and must be studied case by case. It may be hypothesized that it can be put into relation with different issues raised in the preceding sections, such as the level of granularity of the script, the technological-integration options, reasons why students would use the provided technology or the strategy used to present students with this technology. Taking as examples our experiences with the RSC script, different and unexpected uses taking place in relation to the emergent nature of activity can be listed: use of communication functionalities as means of perception for mutual presence or actions; use of a given functionality to edit a result (elaborated via other means) when this functionality and its underlying notions had been designed as means to elaborate the result, thought of as a "support for thinking," and considered as a vector for the targeted learning; means designed to allow editing a result used as a "support for thinking;" change in the way the environment is used due to the evolution of motivations (and, thus, effective activity), for example from « playing the game of the didactic contract and using the platform to meet the teacher's demand » to « deal with urgency and produce the expected result (whatever the means are) »; etc.

Dealing with unpredicted events

Macro-scripts, as a particular kind of pedagogical method, are intrinsically related to open issues that cannot be fully defined or predicted. It is not possible to exhaustively list and consider all the pedagogical parameters of a macro-script situation. As a consequence, when monitoring the script as it is enacted by the students, the teacher often has to manage unexpected events (originating from inside or outside the script), manage requests from the students that will lead him/her to consider script's or technological-setting's modifications, or use a pedagogical opportunity that appears (Dillenbourg & Tchounikine, 2007). As examples, teachers may want or need to modify, at run-time, decisions taken when tuning the script: change the groups because a student drops out of the course or because two conflicting leaders emerge from a group and sterilize interaction; postpone some deadlines in order to deal with external or internal reasons (network failure, bad appreciation of the task difficulty, etc.); change the script structure (change the order of phases, add or remove a phase, merge some tasks, change the argumentation tool because students face problems with it); etc. (Dillenbourg & Tchounikine, 2007). This creates uncertainties related to macro-scripts' enactment, to be taken into account when studying the technological dimensions.

Students' self-organization

In the Computer-Supported Collaborative Work field, *organization* is defined as the meta-level activity that aims at maintaining a more or less stable pattern of cooperative arrangement between people engaged in a collaborative work, which requires the elaboration of an artifactual and/or psychological instrument crystallizing the cooperative activity motives and means (Schmidt, 1990).

Although CSCL scripts' objective is to introduce a structure, macro-scripts may still leave space for some students' self-organization to emerge (Tchounikine, 2007). This space is correlated to the script granularity; almost null in micro-scripts, students' self-organization may become a core issue and require a specific interest in coarse-grained scripts. For instance, in project-based scripts such as RSC, students can and do become active in deconstructing tasks into subtasks, delegating roles or managing time. To some extent, they complete and/or adapt the script. Students' self-organization may however also impact shorter scripts. For instance, a script may fix issues such as grouping students and stating they have 3 hours to achieve a given task (e.g., edit the final grid in ConceptGrid). Such a constraint still leaves open different organizational possibilities: students can decide to spend 1 hour each on the same issue (or on different issues) and then share their thoughts; they can decide to adopt a more structured process and explicitly split the 3 hours into different phases (e.g., brainstorming, elicitation, argumentation and decision); they can use different communication tools with different specificities (e.g., whiteboard or chat); etc.

Student self-organization is not a component of a script, but an abstract object, that emerges from the way students enact the script, and may vary run to run. Following the arguments presented here, some organizational issues are constrained by the script and some others are left open and may take origins in different issues (e.g., individual characteristics of students, social issues within the group such as the emergence of a leader, institutional context or experience or technological setting).

In addition to the obvious fact that students' emergent self-organization should not contradict the script's pedagogical objectives, scripts that leave space for such emerging organization carry a tension between different issues. Macro-scripts suppose a high engagement and a kind of agreement between the teachers and the students and among the students (a kind of "didactical" contract in the sense of Brousseau (1998)): there is an assumption that students will "play the game" and appropriate the script principles to themselves. Easy appropriation has been identified as a criterion for script design (Dillenbourg, 2002). However, when the fact that how people appropriate to themselves and/or develop a shared understanding of a structure is generally related to how much the structure has been collectively constructed and/or refined, scripts (and thus the organization issues they carry) are defined by teachers. Another issue is that when organization is fundamentally a structure that emerges, is unstable and evolves during activity, some script issues carrying organization-related features may be reified by the technological setting, here again with the risk of making technology become a counter-productive constraint. Self-organization is thus another example of uncertainties related to macro-script enactment that it is important to take into account when studying the technological setting.

Implications

How students will perceive the technological setting (and even the script presentation or teachers' explanations) and/or enact the script may be to some extent subject to unpredictability. Moreover, the technological-settings' characteristics may be picked up in different ways by students, who will appropriate them according to their purposes, and in terms of their own current interests or needs. How a student will perceive the script and the technological setting can thus not be *defined*, though it can be *impacted* by the fact it takes part of its origins in the script structure and presentation, and in the technological setting. Other phenomena such as students developing a self-organization or teachers having to deal with unexpected events also participate in macro-script perception and enactment unpredictability. It is important to note that these emergent and sometimes unexpected uses can not simply be addressed by an iterative design process that would, after a certain number of iterations, allow fixing the «standard» use of the technological setting. The same technological setting proposed to similar groups of students may be used very differently according to features such as group dynamics (leader, conflicting leaders, etc.) or the fact (related to different itineraries or individual differences) that the students have different representations of the setting.

The technological setting associated with a macro-script is thus to be studied with respect to how it supports and/or impacts activity in a way that is coherent with both (1) the objective of scripting and (2) the uncertainties related to perception and enactment. These conclusions are in line with more general analyses, such as that of Jones & al. (2006), who "… suggest a flexible approach to design in which designed artifacts are thought of as shells, plastic forms that incline users to some uses in particular but are available to be taken up in a variety of ways and for which the enactment of preferred forms depends upon the relationships developed in relation to learning."

3.4. Discussion

A macro-script can and must be first defined at an abstract, technologically-independent level. When operationalizing it, different design decisions must be made, some of which relate to technology. These technological dimensions can be addressed in various ways, from a very abstract approach, limited to identifying the functionalities necessary to achieve the tasks proposed by the script, to very detailed studies of how technology can participate in structuring student collaboration according to the targeted script objectives, or of how the uncertainties related to perception or enactment of scripts should be taken into account. In order to allow inter-comprehension amongst the different actors involved in the operationalization process, and to facilitate knowledge accumulation, it is thus of core importance to make explicit matters of concern and ways issues, such as the script reification, the way students are presented with the technological setting or the uncertainties related to perception or enactment are thought of. To what extent technological dimensions are considered as means for the supports and constraints targeted by the script can be related to different issues: matter of concern (e.g., approaches focusing on the social dimensions of the script or focusing on the technological conditions of its enactment); level of granularity of the script (the properties of a given tool only become pertinent if they can be put into relation with some detailed principles of the script); setting features (e.g., young children in school or university students acting in an open setting and able to use whatever means they prefer); etc. We do not claim that using the technological setting to reify script principles is necessarily the best approach, but that the role and expectations related to the technological setting should be made explicit.

From the point of view of collaboration, macro-scripts witness the tension between instructional design and socio-cultural approaches (Dillenbourg & Tchounikine, 2007). The same kind of tension appears at the level of how technology can be used to participate in structuring the process. Within CSCL scripts, focus is not on the task output (e.g., the common document to be produced by students) but on the process (e.g., what happened during the elaboration of the document, such as arguments, exchanges or common understanding elaboration). In some situations, forcing students to use a given technology can become counter-productive. At the same time, if a communication tool or a workflow has been designed or chosen because its use is supposed to suggest or support a given behavior, how this tool is used is an issue. This differs from settings where what is important is the produced output, whatever the means and the process are.

When considering technology as a way to continue scripting or, at least, as a dimension that has an impact on student enactment of the script, the coherence between the features carried out by the technological setting and the script's overall objective is an important issue that must be addressed explicitly. First, in addition to the way the teacher explains what is to be done and how, students may also perceive the script from the way the technological setting reifies it (this can be asynchronously, if the script is orally or textually proposed first and the technological setting introduced later on, or synchronously, if the script description is embedded in the technological setting). The script and the technological setting are two sources of information, support and constraints that are perceived in an interrelated way by students, and both influence the students' perception and understanding of the script. The technological setting and the script must thus be studied in order to have (as far as can be predicted and using iterative analyses) a coherent and articulated impact on the students' understanding and perception. A minimal requirement is that technological settings must provide students with the necessary technological means in a way that is not incoherent with the pedagogic intentions, unnecessarily constraining or confusing. Second, technology plays a role related to the type of behaviour it allows, suggests, supports or makes impossible. When considering the objective of scripting student processes, different (non exclusive) means can thus be used from which (1) the script (tasks and subtasks, sequencing, constraints, etc.), (2) the way the teacher participates in the orchestration of the script (providing directions, controlling access to resources, orchestrating and regulating students' activities, etc.), and (3) the technology-related design decisions used as a way to support students and/or regulate their process. Such decisions can be related to the properties of the functionalities and tools presented to students and/or the way they are presented (e.g. features related to their integration within articulated interfaces, their conditional access using data-flow or workflow processes or to what extent students are provided with flexibility).

This analysis brings us to the conclusion that when considering the operationalization of macro-scripts, the refinement and tuning of the script and the technological setting should be analyzed in an interrelated manner and thought of as articulated resources. This requires a general conceptualization-model that helps in making salient the dimensions to be taken into consideration. We propose such a model in the next section.

Acknowledging that macro-script operationalization involves different intertwined dimensions (e.g. social protocols or technology issues) and has to deal with uncertainties suggests consideration of the notion of *indirect design* (Jones & al, 2006). Within CSCL macro-scripts, indirect design captures the idea that defining the script and the technological setting features and properties must be thought of as means to *influence* student activity, and this activity (and the impact of the design issues) must be taken into consideration as it happens, and not as it was predicted by designers. Addressing the elaboration of the script + technological setting couple is not a one-shot problem. During first elaboration a certain number of issues can be defined, but others may only be addressed as hypotheses (e.g., flexibility requirements or student perception). Design must thus be addressed iteratively, and benefit from longitudinal studies confronting design models and students' effective interaction patterns. At this level, an important issue is that experiences should be repeated, in order to attenuate the bias introduced by the problems that any user encounters when presented with new software. And the inherent unpredictability of some issues must be acknowledged.

4. SPAIRD: an operationalization-oriented conceptualization of the relations between macro-scripts and technological settings

4.1. Objective and issues

Seen from the perspective of the design of technological settings, macro-script operationalization can be viewed as follows. Macro-scripts are first described at an abstract and technology-independent level. They are based on explicit pedagogic principles that define (more or less precisely) foundations, principles and constraints for instantiating the script in a given setting, and make the according design decisions (defining the pre and post activities, the groups' composition, the tuning of the different phases, the way the script is presented to students, etc.) from which the design decisions related to the technological setting. In order to enact the script and achieve their tasks and subtasks, students are presented with a technological setting. The role of this technological setting can go from just providing functionalities that allow achieving the task to participating in the structuring of the student process. It can be more or less integrated, and in different ways. It may have an influence on how the script is perceived by students, and this dimension must thus be taken into account in order to avoid contradictions and/or complement suggesting the targeted behavior. In certain cases, students may be given certain latitude, as a pedagogical strategy and/or as a way to acknowledge script enactment uncertainties. Consequently, macro-scripts and/or technological settings should be to some extent flexible. Different (intertwined) reasons may encourage teachers/designers to consider flexibility, for example: an explicit pedagogical choice of providing self-service conditions; the fact that the setting limits the way constrained conditions can be created (e.g., distant autonomous students); the objective of proposing suggestions that are thought of as shaping collaborative interactions whilst avoiding over-constraining the setting and student collaboration (i.e., emphasizing the fact students play the script within a context defined by the script's constraints and the technological setting, rather than the fact the technological setting plays the script and prompts student actions).

Within this perspective, our work addresses the research question: *What dimensions should be put on the worktable when considering the technological dimensions of macro-script operationalization?* The proposed contribution is a conceptualization model (called SPAIRD) that disentangles different dimensions/notions involved in macro-script operationalization, and allows making explicit the issues and features to be considered and the matters of concern. This is proposed as a basis to facilitate how (non-technical) educators and computer scientists can collaborate to address script operationalization and make detailed operationalization decisions when selecting, customizing or constructing the script's technological setting. It is also a basis from which to identify/develop specific tools (e.g., specific modeling languages) addressing a specific given issue.

The proposed model aims at acting as a descriptive and informative framework for the design and architectural structuring of technical support systems, taking into account the general analysis presented in Section 3. From this perspective, it is an intermediate construction between (1) works that consider as an entry point educational issues (e.g., works such as Kirschner & al. (2004) or Strijbos & al. (2004), cf. introduction) and (2) works that focus on a given technology/framework or a given precise modeling language. The addressed level is thus that of a conceptual framework, that provides a kind of pre-structured map for relating pedagogical issues and issues of technology design. We believe this has a value in itself, as a tool for thinking, and as an intermediate work towards the specification and implementation of precise modeling languages targeting advanced support functions and/or design methodologies (cf. §5.1).

As macro-scripts are associated with platforms rather than standalone tools we will use the wording *platform* with the general meaning of a computer-based system that provides/integrates different functionalities or tools. This can correspond to an all-in-one system (i.e., a system that provides different functionalities within a dedicated integrated interface) or a platform in the usual sense (i.e., a set of components or tools made accessible from a common interface).

4.2. General presentation

The analysis presented in Section 3 made salient the fact that the operationalization of a macro-script requires considering different interrelated perspectives. Operationalizing such scripts must therefore be thought of as a complex problem, i.e., a problem that involves different issues interacting one with the other in a systemic way, and that cannot be reduced to any of these issues (Lemoigne, 1990). Following the theoretical background of complex-systems, this kind of problem must be addressed on the basis of multiple points-of-view and models denoting different perspectives (including partially-redundant perspectives) on the considered objects.

Within this perspective, we propose a model called SPAIRD (for Script-PlAtform Indirect Rational Design). The objective of this model is to propose a general conceptualization (a general picture) that helps designers in defining what is to be taken into consideration when selecting, customizing or designing macro-script technological settings in order to (1) avoid problems and constraints arising from technology and/or (2) attempt to use the script and the technological setting as coherent articulated vectors targeting the students' expected behavior. As such, it is a descriptive and informative framework for the design and architectural structuring of technical support systems. It makes salient a number of dimensions, notions and issues in a way that is complementary with, on one side, more educationally or psychologically-oriented models of scripts, and, on the other side, operational languages and technologies. In section 5 we explore how this model leads to addressing technological setting implementation and processing principles.

SPAIRD disentangles the different following notions:

1. Structural model of the script
2. Implementation oriented model of the script
3. Platform specification
4. Student oriented models of the script and platform
5. Design rationale (learning hypothesis, pedagogic principles, design decisions)

The first four notions denote different but non-independent *viewpoints*. In software engineering, a viewpoint is a technique for abstraction using a selected set of concepts and rules in order to focus on particular concerns and build viewpoint models, i.e., a representation from the perspective of the chosen viewpoint (MDA, 2003). SPAIRD addresses design issues and considers perspectives and models related to the script components (structural model), the script implementation, the platform specification and the presentation of the script to students. Similar to approaches such as UML (UML, 2006), models are considered here as means to (1) make designers and analysts consider a given issue by the fact that this issue is outlined and (2) help them to study the issue by proposing a set of notions and/or a modeling language. As a conceptualization model, SPAIRD addresses the first dimension (outlining issues) and not the second (providing a specific language for each model), which is a different question, and for which different existing works can be reused. The four models outlined by SPAIRD are not to be considered one after the other but simultaneously and in a systemic way, although some models (typically, the structural model) will be considered but not necessarily closed before some others. The fifth notion (design rationale) denotes the rationale behind the elaboration of the four preceding models.

Figure 1 presents a general overview of the proposed model. Block A corresponds to the design rationale, i.e., the principles and decisions that underlie the script modeling. Block B corresponds to the different models of the script to be elaborated. Block C denotes the platform issues. Block D corresponds to the students' perspective. We focus on the script and platform issues, thus targeting design models and not student-behavior models. Whilst not deeply examined in this article, block D's presence is meant to remind one that the script and platform only define a set of propositions whose perception and use by students are linked to many other different issues, such as individual, social or institutional issues.

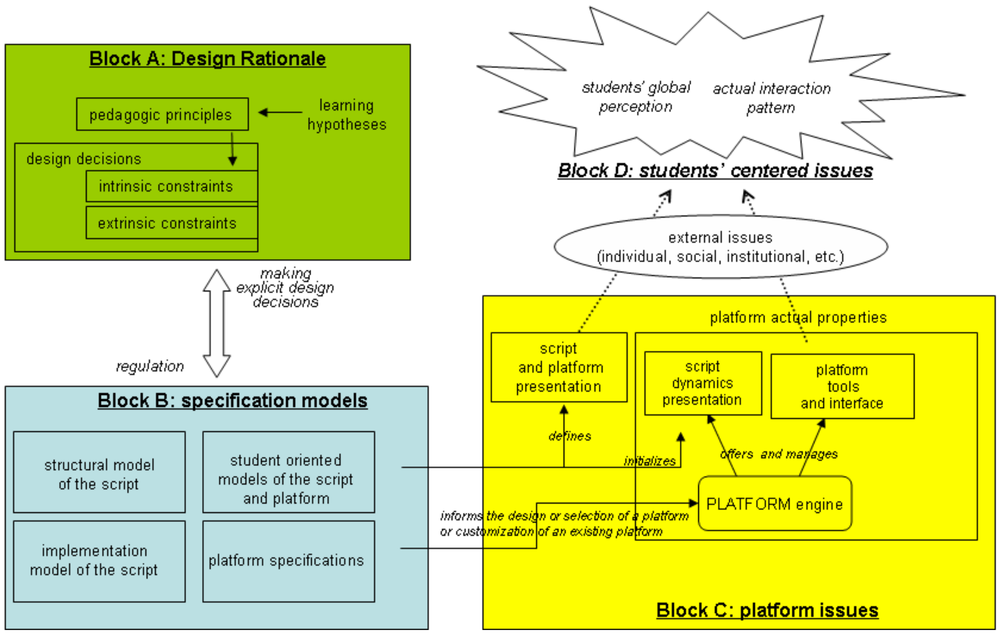
Block A: Design Rationale
pedagogic principles
learning hypotheses
design decisions
intrinsic constraints
extrinsic constraints
making explicit design decisions
regulation
Block B: specification models
structural model of the script
student oriented models of the script and platform
implementation model of the script
platform specifications
students' global perception
actual interaction pattern
Block D: students' centered issues
external issues (individual, social, institutional, etc.)
platform actual properties
script and platform presentation
script dynamics presentation
platform tools and interface
defines
initializes
offers and manages
PLATFORM engine
informs the design or selection of a platform or customization of an existing platform
Block C: platform issues

Figure 1. The SPAIRD model

4.3. The sub-models

As stated previously, our focus is on arguing for the importance of making these models explicit, and not on proposing specific modeling languages. Different existing languages can be used for some of these models. Elaborating a set of coherent holistic modeling languages for all the different outlined models, if necessary, is part of a longer research agenda. Therefore, we will remain at a conceptual level and only introduce a few examples with links to existing modeling languages or interesting related approaches for the sake of clarification.

Structural model of the script

The *structural model* is a description of the components of a script. This point of view corresponds to an analysis in terms of *what* is to be defined when designing a script, i.e., a set of notions and, possibly, relations linking these notions.

```
<Script> ::= <pre structuring activities> <Phase>+ <post structuring activities>

<Phase> ::= (<Task> <Group> <Mode> <Task oriented tools> <Interaction mode> <Timing>)+

<Task> ::= <input> <activity> <output>

<Group> ::= {individual}+

<Mode> ::= individual | collective | collaborative

<Task oriented tools> ::= specific tools required to perform the task, if any

<Interaction mode> ::= face to face | <communication tool>+

<Communication tools> ::= basic chat | basic forum | basic email | basic electronic whiteboard |
                          file exchange zone | specific tool

<Timing> ::= duration | output delivery deadline
```

"::=" stands for "is decomposed in", "|" stands for "or", "*" stands for "zero or several" and "+" for "one or several"

Figure 2. A simple structural meta-model (using a BNF-like syntax)

In order to exemplify the idea whilst not going into the details of a specific language, we present in Figure 2 a (voluntarily trivial) example of a structural meta-model, i.e., a language that denotes script components. Different languages addressing this structural dimension have been proposed in literature. For instance, Kobbe & al. (2007) identify groups, participants, roles, activities and resources as the basic components of a script, dissociating these components from the script mechanisms (group formation, component distribution and sequencing, that we refer to as the implementation model). Hernández-Leo & al. (2006) describe how the notions underlying the general educational modeling language IMS-LD (IMS-LD, 2003) can be used to model CSCL script structure, and LDL (Ferraris & al., 2007) is presented as an alternative to LD for collaborative settings.

A structural description of a script does not denote dynamic issues such as data-flow, tasks sequencing, group-formation or role-attribution mechanisms. It however already allows denoting part of the script's characteristics. For instance, considering the flexibility issue, phases are unlikely to be flexible since they define the script's general structure. Similarly, whether a task is individual or collective or the interaction mode (face-to-face or via communication tools) is often determined by the structure, unlike issues such as timing. In some scripts, the way groups are composed and/or the role attribution can be left open or partially open to students. For instance, in a script such as ConceptGrid, students can be allowed to compose groups or modify them with respect to the constraint stating that groups must be composed of individuals mastering different knowledge. In the context of coarse-grained scripts such as RSC, the precise definition of the input and output of the different tasks composing a phase can be left open to students when setting the script and/or while running it, for instance allowing or even suggesting to students that they can again split the proposed subtasks into finer-grained subtasks. According to the way the technological setting is thought of, task oriented tools and communication tools can become typical candidates for flexibility if one considers that students should be contextually able to choose the tools they want. As one can see from these examples, the structural model puts some issues to be considered on the worktable, but is not sufficient to address them: other analysis dimensions are required.

Implementation model of the script

The *implementation model* of the script is the description of how the script is to be put into practice. While the structural view states what the script components are, the implementation description states what constraints rule them and how they should be orchestrated. An analogy can be made with object-oriented software engineering and the difference between modeling the world (the objects and their characteristics) and modeling how the objects and characteristics are used to implement a given functionality (the collaboration model).

The implementation model of the script describes issues such as: group-formation policies and dynamics; task sequencing and articulation; the dataflow/workflow ruling the access to individual and collective data and/or to functionalities/tools; etc. Such implementation issues can be described at different levels. Informal representations may be sufficient is some cases, for example when the script is orchestrated by the teacher. Conversely, when the computer-system is meant to run some of these issues, the modeling must go into detail, and a modeling language that is associated with an operational semantics is required. The language proposed in Miao & al. (2005) is an example of an operational language that allows representing implementation models (in fact, representing the structural models and the implementation model at the same time). Another example is proposed in (Haake & Pfister, 2007). Such languages propose advanced constructions to model the <activity> notion as denoted in Figure 2.

Considering the implementation model, an important dimension is to dissociate scripts' *intrinsic constraints* and *extrinsic constraints* as proposed in Dillenbourg & Tchounikine (2007). Intrinsic constraints are bound to the script's core mechanisms, e.g., during the collaborative phase individuals must manage different knowledge. Extrinsic constraints are bound to contextual factors, e.g., individuals can be conducted to master prerequisite knowledge following different approaches; different pre/post activities can be added in order to trigger the core mechanisms; group compositions can take different issues into consideration; etc. Extrinsic constraints define the space for flexibility, i.e., the space within which a script should be modifiable by teachers and/or students because the related decisions result from arbitrary or practical choices. Within this perspective, 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, if targeting a situation where two students confronted with different initial knowledge should interact, allowing free pairing would violate an intrinsic constraint, and would break the link between the script and the underlying learning principle; conversely, the run-time modification of groups should be allowed and managed (e.g., insuring data coherence) if modifying a group is a possible option for the teacher or the students.

Although structural and implementation models are interrelated and in some cases partly redundant, we believe their dissociation and an explicit description of the implementation model are important to avoid indirect strategies and/or implicitness. Let us consider the following crossing mechanism: in phase1, two groups (or individuals) G1 and G2 work on an issue, and each produce an output; then, in phase2, each group is asked to elaborate on the content produced by the other group. Taking the structural modeling language proposed in Figure 2, a workflow issue such as the fact that G1 output of phase1 is the input for phase2 of G2 (and vice versa) can be considered as a straightforward conclusion from the description of the phases. However, this is not the case for all workflow issues. For instance, the fact that during phase1 the G1 and G2 subgroups must not be aware of each other's productions as this would spoil phase2 is not explicit in the structural description. Less trivial examples include detailed sequencing (e.g., task parallelism or task conditional synchronization) or data accessibility (e.g., stating when data and/or functionalities should become available, for instance making what is necessary for the achievement of a task accessible when, and only when, the preceding task is over) cannot be easily described by a structural description.

As structural and implementation models are interrelated, they are addressed as such in some works such as Miao & al. (2005). Although dissociating them may appear artificial in some cases, in some others it has the virtue of leading to an analysis of script constraints and of their nature (intrinsic constraints, explicit constraints) in a process that is disentangled from the structural characteristics of the script. From our perspective, this helps in analyzing constraints for what they are, and not via the way they impact structures or are carried by structures. For instance, applying a crossing mechanism over groups (i.e., groups are reorganized when skipping from phase1 to phase2) can be denoted by a structural description of phases: during phase1, G1 = (Lucy, Jack) and G2 = (Bill, Connie); during phase2, G1 =

(Lucy, Bill) and G2 = (Jack, Connie). This principle would however better be denoted by the underlying abstract principle “during phase2 groups must be constituted with students that were in different groups during phase 1”. Making such principles explicit is a core issue, in particular for (1) studying how they will be taken into account in the different models and not only the structural model and (2) how they can be managed dynamically to comply with flexibility issues, or be overruled. Computer-inspired languages such as the one proposed by Miao & al. (2005) or Haake & Pfister (2007) allow representing and implementing such constraints (as said before, given our research perspective, we will not elaborate here on the fact that these languages’ complexity may make them difficult to use for practitioners; a discussion on this issue can be found in Harrer & Malzahn (2006)).

Platform specification

The *platform specification* is the set of technical specifications (in the computer-science sense) that the technological setting must comply with. The platform specification is meant to make the technical design decisions drawn from the structural and implementation models explicit.

The form of specification that is required greatly depends on the adopted technological approach. If the approach just addresses the objective of providing students with tools, the platform specification is limited to something like “make a chat and a file-exchange system available.” In such a case, it is more or less redundant with the structural model. If the approach considers in more detail the properties and/or integration of the functionalities to be proposed, it is necessary to specify them: data to be represented and manipulated; data-flow and data-access constraints; functionalities; workflow; user interfaces; etc. The way the platform specification is described is related to the computer-based system it will be deployed on, e.g., a generic platform or an *ad hoc* system (cf. §5.1). In the latter case, this requires computer scientists to build these specifications, using computer science methods and techniques such as the one provided by the Unified Modeling Language UML (UML, 2006), and then build the corresponding system.

Coming back to the affordance notion, the platform specification specifies the *actual properties* of the platform, as an artifact that can be unambiguously described. Defining these properties is the prerequisite step of an iterative and experience-based process that must also take *perceived properties* and *effective usages* into account.

Student-oriented models of the script and platform

The *student-oriented models* of the script correspond to the dedicated information the students are presented with in order for them to understand the script and the platform. This is related to the fact that, as noted in section 3, both contribute, in an interrelated way, to the perception of the script and its associated technological setting.

Considering the script, two dimensions can be dissociated. The *script presentation* corresponds to a description of what students are supposed to do. This can be an oral and/or written presentation, presented separately from the platform (via documents or orally) or embedded in the platform. It can be defined according to different strategies, e.g., providing a comprehensive view of the script or presenting phase *n* when phase *n-1* is over. The *script dynamics presentation* corresponds to a description of what can be understood and represented within the platform of the way the script is enacted by the students. This presentation can be constructed from an automatic analysis of the way students use the platform (using logs and student action analyses) and/or data made explicit by students (e.g., through advancement declaration) or teachers; it can be limited to some advancement information or extended to advanced awareness issues. This is a dynamic issue, which can be used by students to know where they are (and by teachers to know where students are). These two representations can be merged into a single one such as a global “visualization of the script” initialized by the script presentation and then denoting its dynamics (Berger & al, 2001). Such advanced dynamic issues generally require specific platform specifications, using usage track analysis models and their transformation towards students’ understandable representations in the script’s dynamic presentation. In Figure 1 we have dissociated the script dynamics presentation from the platform tools and interface, but this is essentially for the sake of clarity. Finally, the *platform presentation* corresponds to a description of the technological means that are proposed to the students.

Script and platform presentations can be thought of as what is told to the students in terms of “what” and “how”, on the basis of the structural, implementation and specification models. We have emphasized the importance of dissociating these issues from a conceptual and design point of view. However, when considering instructions and explanations for students, it can be (according to the setting) preferable and/or easier to dissociate these two dimensions or, on the contrary, present them together and in an interrelated way.

Design rationale

From a general point of view, a design rationale is a representation of the reasoning behind the design of an artifact. Considering the design rationale promotes making explicit the decisions to be taken, the possible alternatives and the reasons for the one chosen. This also helps in accumulating knowledge reusable for settings with similar rationales.

In the context of the SPAIRD model, we refer to the script’s *design rationale* as the principles that underlie the script and its operationalization. Of course, not all principles can be, nor need be, made explicit. As said previously, the point is to select a set of concepts and rules in order to focus on particular concerns (models capture what we can/want to capture from a certain perspective; models are not reality).

The design rationale of a script can be addressed at different levels of abstraction. We propose to dissociate learning hypotheses, pedagogic principles and design decisions, going from abstract principles to operational decisions. The *learning hypotheses* are the considered abstract general hypotheses about how humans learn that form the base of the script. For instance, the learning hypothesis of the Jigsaw family of scripts could be formulated: "students confronted with a problem they cannot solve individually but can solve collectively by sharing knowledge can learn one from each other." Making this issue explicit is useful for keeping in mind the overall reference; it is however not operational. The *pedagogic principles* of the script are the principles that originate from the learning hypotheses and define the spirit of the script. For instance, for the Jigsaw family scripts, it could be formulated: "the learning situation involves *n* students S1 … Sn; the *n* students have to achieve Task T; T requires some knowledge that none of the Si students master; each Si student has some knowledge that is useful to achieve T; the knowledge mastered by the *n* students together allows the achievement of the task T." The same learning hypotheses can underline different scripts corresponding to different pedagogic principles, and the same pedagogic principles can themselves be turned into different subclasses or script abstract schemas (e.g., Concept-Grid is but a subclass of Jigsaw) that can themselves be turned into different instantiated scripts (Kobbe & al., 2007). Pedagogic principles define the core principles which, if not respected, would make the script not rely on the learning hypothesis any more. In order to turn these concepts into effective scripts, a set of *design decisions* must be made and documented. Design decisions denote the rationale underlying the decisions that have been made while defining the structural, implementation, specification and student-oriented models.

An intrinsic drawback of multi-points-of-view modeling is to manage coherence. Keeping an explicit trace of design decisions (i.e., the alternatives, the decision and its justification) does not ensure coherence, but is a sine qua non condition (and, as will be advocated in Section 5, a potential means for controlling flexibility issues). One option would be to distribute design decisions over the different models. However, many principles and decisions impact different issues in different models, and grouping design decisions may facilitate the overall coherence management. This remains, however to be studied.

When considering design decisions, dissociating intrinsic and extrinsic constraints is an important issue. For instance, within a script that uses a crossing mechanism over groups, there is an intrinsic constraint to be respected for the creation of the groups: if phase1 aims at making Lucy and Jack familiar with theory A and Connie and Bill familiar with theory B, phase2 groups must be composed with one student from each of phase1 groups. This issue must be made explicit. However, whether phase2 groups are (Lucy, Bill) and (Jack, Connie) or (Jack, Bill) and (Lucy, Connie) is contingent, and can be addressed accordingly, e.g., arbitrarily or according to some extrinsic constraint related to the gender composition of groups. It should be noted that design decisions can be made explicit at different degrees of abstraction. Stating that "Lucy and Bill must be paired" is a low-level constraint. Stating that "pairs must be constituted of students exposed to different knowledge during the preceding phase" is a more abstract constraint that allows more flexibility when running the script if it appears that Lucy and Bill hate each other. An even more abstracted principle would be "pairs must be constituted of students mastering different knowledge." This would allow more flexibility when running the script if it appears that, although Lucy worked out theory A, she did not develop enough competencies to have knowledge generative interactions with Bill but, as Connie had some quite consistent pre-existent knowledge about A, it is finally preferable to group Connie and Bill. Abstract principles offer more latitude for flexibility. However, they are also more prone to be difficult to represent in an operational way.

Student-centered issues (block D)

The models we have outlined are useful to support design. Let us however recall that script and platform only define propositions whose perception and enactment by students are linked to many other issues. In Figure 1 (block D) we have mentioned two rough notions that reflect our concerns: the *students' global perception* 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 (Dillenbourg & Tchounikine, 2007)). This is sufficient for our matter of concern, but not *per se*.

4.4. Using SPAIRD to make explicit and guide design decisions

SPAIRD is a conceptualization model that makes salient a number of notions and issues that designers and analysts should put on the worktable when studying macro-script operationalization. The dissociation of different models disentangles issues that are often kept implicit and/or mixed, such as the script's different dimensions, the interrelations of the script and the technological setting or the vectors that can be used to influence student perception and enactment. The design rationale helps in making explicit the rationale behind the decisions.

Such a proposition cannot be evaluated through empirical studies or prototyping but, as a conceptualization that aims at supporting design, it can be questioned in respect to its usefulness. In order to show this usefulness, we present here below examples of how SPAIRD allows outlining different issues and drawing general propositions with respect to the couple script + platform.

At a general level, SPAIRD helps in outlining issues to be dissociated and documented. For instance, general *pedagogic principles* and *design decisions* should be disentangled. These *design decisions* should be as far as possible dissociated into *intrinsic constraints* and *extrinsic constraints*. Any pedagogic issues that are addressed opportunistically as a second objective, e.g., learning to work in groups, could be related to *extrinsic constraints*. *Pedagogic principles, intrinsic constraints* and *extrinsic constraints* should be represented at an abstract level, dissociating abstract principles and context-related knowledge. All the decisions underlying the *structural model, implementation model, platform specification* and *student-oriented models* should be made explicit, i.e., documented and justified. The *structural model* and *implementation model* elaboration should carefully dissociate the script issues that correspond to "what" (*structural model* related issues) and "how" (*implementation model,* and then *student-oriented models* or *platform specification* issues).

When considering script/platform coherence issues, SPAIRD helps in outlining what issues relate one to the other. For instance, *pedagogic principles* must be coherent with the *learning hypotheses*. The *intrinsic* and *extrinsic constraints* must be coherent with the *pedagogic principles* (and thus with the *learning hypotheses*). The *structural model, implementation model, platform specification, student oriented models, script and platform presentation* and *platform actual properties* (which includes the *script dynamics presentation* and the *platform tools and interface*) must be kept coherent with *intrinsic* and *extrinsic constraints*, and with each other. The *platform tools and interface* should not impose issues that are or could contradict the *structural model, implementation model* or *student-oriented models*. The *script dynamics presentation* should be updated in real time in order to maintain coherence with the *actual interaction pattern*.

When considering the completeness of the modelling and how the different features form a self-sufficient framework for students, SPAIRD helps in listing issues to be checked. For instance, it can be an objective that the *script and platform presentation* and the *platform actual properties* should denote all the *structural model* and *implementation model* issues that are necessary for students to understand what they are supposed to do and to do it; another option is to consider these issues with respect to some planned additional regulation by the teacher or the system. Features such as the set of differences between the *platform tools and interface* and the *platform specification* should be considered. This can be addressed by human analysis or on the basis of a model of the platform. For instance, when using a customizable platform, it can be an objective to avoid functionalities that are not useful for the script, in order to limit students' cognitive charge and potential disorientations or unnecessary browsing; another option is to keep the same general interface from script to script, in order to facilitate platform appropriation.

When considering how script reification may be used to contribute, together with the *student oriented models*, to suggest desired behaviors by influencing student perception and guidance, SPAIRD helps in denoting notions and issues to be considered. For instance, the platform must be studied with respect to the *platform specification* and to the *students' global perception* and *actual interaction pattern*, using psychological and usage experiments and/or accumulated experience (possibly, going into subclasses such as students' profiles). This can cause the *platform specification* and then the platform to be modified, or flexibility to be introduced. The platform should reify *platform specification* issues that originate (via the *structural* and *implementation models*) in *intrinsic constraints*, i.e., the platform should reify the core mechanisms of the script. Conversely, the platform should not reify *platform specification* issues that originate in *extrinsic constraints*, i.e., the platform should not reify any contingent issue. The platform reification of *intrinsic constraints* should as far as possible be implemented at an abstract level, i.e., implementing the principle and not its application in a given context. Consequently, the platform should propose mechanisms to instantiate the abstract principles according to the particular context.

When considering flexibility issues, SPAIRD helps in denoting what components can be considered as candidates for flexibility, and within which constraints. For instance, we have outlined that a *structural model* denotes different components, some of which can be made flexible. Making explicit the *implementation model* and the *student-oriented models* helps in identifying what can be decided or adapted at runtime by the students or teachers whilst remaining coherent with the *intrinsic constraints*. The platform should then allow and support (i.e., provide functionalities for) the targeted flexibility issues. The platform's adaptable issues must be highlighted in order for students to be aware of them via the *script and platform presentation*, and the *platform actual properties*. This should be checked by questioning the *students' global perception* and the effective interaction patterns. The platform should also avoid fixing issues identified as potentially flexible, e.g., the platform should not hard-code data-accessibility rules if roles can be questioned at runtime.

When considering student self-organization issues and organization-related flexibility, SPAIRD helps in studying the crucial issue of how much the emergent organization (if any) is coherent with *intrinsic constraints*. It also helps in working out how, if at all, suggested organizational issues can be studied in the *structural model*, the *implementation model* and the *student-oriented models*, and the corresponding design decisions recorded. In such a case, the *script and platform presentation* should suggest an organization whilst making clear how flexible it is, and what the students' latitude is.

When considering script enactment, SPAIRD helps by providing notions that allow going further than a simple comparison of the *actual interaction pattern* with "the script" as a broad notion. For instance, the *actual interaction pattern* can be analyzed according to different dimensions, such as the script *structural model* (e.g., phases and tasks), the expected behavior (as denoted by the *implementation model* and *student-oriented models*) or the expected use of the platform (*platform specification, platform actual properties*). This analysis can take into consideration the rationale that underlies these issues (*learning hypotheses*, *pedagogic principles*, *intrinsic* and *extrinsic constraints*). This allows, for instance, studying to what extent, and why, whether the fact that things went coherently or differently from the script is positive, negative or neutral with respect to the *design rationale*. Hypotheses related to the *actual interaction pattern* can be stated, for instance the influence of different issues (*script and platform presentation*, *script dynamics presentation* or *platform tools and interface*) on *students' global perception* and the *actual interaction pattern*. For example, recurrent *students' global perceptions* that do not originate from any design decisions should be made explicit, analyzed and questioned. Accordingly, hypotheses related to potential parameters that can be tuned in order to influence scripts' perception and enactment can be stated and tested. Whilst SPAIRD is not meant to analyze the interaction per se, it is potentially a useful contribution to analyze the interaction with respect to the design.

To conclude this section, let us recall that the set of propositions here are not proposed as a comprehensive set of guidelines, and can be questioned. They do however illustrate that (1) SPAIRD conceptualization is a useful substratum for creating a general picture that makes salient notions that are often mixed or difficult to refer to, and (2) addressing macro-script operationalization issues requires making references to different notions originating from different points of view, both educationally and technologically orientated.

5. A general analysis of technological settings for CSCL macro-script operationalization

In this section we first analyze different current technological approaches to macro-script operationalization with respect to the issues that arose in this work. It should be noted that these different approaches do not all consider the same objectives or issues. We analyze them as possible technological settings for macro-scripts with respect to how they are meant to be used (not elaborating, for instance, on the fact that a generic tool meant to be used as such can of course be modified by a computer-scientist to match a given different specification). We then analyze the interest and difficulties of the current evolution towards model-driven approaches, and the flexibility issue. We finally present our own view of future platforms as tailorable model-based script-engines as a direction for future works.

5.1. Prototypical architectures/approaches used for macro-scripts' operationalization

At a general level, CSCL settings can be operationalized using standard technologies, e.g., workflows. The motivation for using specific software is that standard software is often considered not suitable for students and/or the considered pedagogical setting, i.e., software designed for working processes does not present the suitable properties for learning contexts. We differentiate here different prototypical architectures/approaches that can be used for macro-script operationalization: Learning Management Systems, Content Management Systems, platform generators, operationalization languages and script-specific platforms.

Learning Management System (LMS) are general purpose platforms. As such, they do not propose any feature specific to macro-script operationalization. They do, however, allow implementing the self-service approach, i.e., providing students with open access to the functionalities required by the script. LMSs natively propose generic functionalities, such as chat, email, shared agenda or file exchange zone, and most of them allow external tools to be made available from their interface. Within such approaches, due to the fact LMSs are generic general purpose platforms, students are offered a script-independent interface: instructions, tools and resources are made accessible, but the script remains diffuse. With respect to SPAIRD, this can be interpreted as limiting the technological dimensions of operationalization to (1) selecting a ready-to-use platform that provides the functionalities or tools identified in the *structural model* and (2) providing students with a *script and platform presentation*. Some LMSs natively propose and/or can be enhanced with general awareness functionalities, such as indicating information related to students' connexions or recent deposit of files or messages. This however remains a limited approach to the *script dynamic presentation*.

Content Management Systems (CMS) are platforms similar to classic LMSs in their generic character but are (as a more recent generation of such systems) generally designed as an integration of modules. This natively allows customization by integrating or withdrawing modules taken from a set of already available ones and/or others designed or modified for the considered context. A generic but often used in educational settings CMS is Zope/Plone (Zope, 2006; Plone, 2006). The dissociation between the basic Web service (Zope) and the toolkit (Plone) allows an easy creation of customized platform instances. This customization can address the tools issue (selecting tools according to the script) and/or some interface issues. Typically, a generic CMS interface is defined as a default organization of predefined constructs such as tabs or folders. Such an interface can be customized by modifying this organization according to the script: adapting tabs to denote groups/subgroups and or phases/tasks; adopting a specific folder organization to create work zones composed of modules such as a file-exchange zone or communication tools; etc. With respect to SPAIRD, using such a CMS can be interpreted as implementing by hand the *platform specification*, the *script and platform presentations* and the *script dynamic presentation* with means that correspond to the CMS customization possibilities. For instance, Zope/Plone proposes possibilities to adapt tabs and folders and associate them with access to tools, or to manage document life-cycles (e.g., how a document goes from "private" to "public"). The interesting issue is that this customization is technically easy as it takes place via an administration interface. However, as these customization possibilities are limited, using a customizable CMS does not guarantee that the aspects of the script that designers want to reify can be reified (except, of course, by going into the code).

Platform generators are dedicated systems that allow generating or adapting a platform via an education-oriented description of the script (as opposed to a technically-oriented description). As a first example, in the Collage-Gridcole approach, teachers are presented with a pattern-based authoring tool that facilitates script modelling. From this description, a platform is generated in the form of an interface to grid services (i.e., services provided by web services) that correspond to the tools required by the script (Hernandez-Leo & al., 2006; Bote-Lorenzo & al., 2004). As another example, the Bricolage approach allows the description of scripts using a graphical description, this description being then used to automatically adapt the code of an existing platform on the basis of a connection between (1) the model of the script and (2) the architectural model of the used platform (Caron & al., 2005). With respect to SPAIRD, these approaches can be interpreted as providing teachers/designers with an authoring interface corresponding to SPAIRD specification models (*structural model*, *implementation model*, *platform specification* and *student-oriented models*; in these works, however, these models are not clearly disentangled). To what extent the desired properties of the platform can be obtained via these approaches is related to the focus and level of precision of the modelling languages underlining these authoring tools. In the Collage-Gridcole approach, emphasis is on authoring and facilitating teachers' work. The approach is based on the use of the educational modelling-language IMS-LD (Hernandez-Leo & al., 2004). LD allows listing the functionalities requested by a script. This first step is somewhat similar to selecting what components should be made available within a CMS, with however the central advantage of doing this via teacher-dedicated authoring tools. LD also allows representing some aspects of the SPAIRD *implementation model*, such as the flow of activities or the role distribution, though not in a straightforward and complete way (Hernandez-Leo & al., 2007; Miao & al., 2005). This allows running the script with a LD-compliant engine (cf. infra), such as CopperCore (CopperCore, 2007). With respect to SPAIRD, this corresponds to a *platform specification* that lists the requested functionalities (that will be presented to students in a self-service approach) and/or an *implementation model* limited, however, to issues that can be represented with LD. Another example of such an approach is the LAMS platform (LAMS, 2007) that provides teachers with a graphical editor allowing them to define (very simple) scripts by connecting predefined tools, and then generating from this description a specific interface for the students.

Operationalization languages are languages associated with an operational semantics, which allows simulating or running the represented script on a computer. Such languages can be used as a way to implement the details of how scripts should be orchestrated. This includes actions to be achieved before the script is enacted (e.g., composing groups) and during script enactment (e.g., orchestrating phases, re-organizing groups or implementing a workflow). With respect to SPAIRD, this corresponds to the possibility of describing and running detailed *implementation models* of scripts. An analysis of different modeling languages that can be used to represent such issues (finite automata, statecharts, activity diagrams, Petri nets, BPEL4WS, IMS-LD and Mot+) is proposed in Harrer & Malzahn (2006). The authors emphasize 4 important dimensions of such languages: (1) the fact that they are familiar and easy to use for practitioners; (2) the fact they propose a graphical representation (here again, this is related to facilitating use); (3) the fact that the language allows modeling at different level of granularity, from the general orchestration of phases to the detailed modeling of interaction patterns; and finally (4) the fact that the language is associated with an operational semantics. Within the perspective adopted in our work, the first three dimensions can be related to producing models that can serve as intermediation objects for multidisciplinary design teams (i.e., boundary objects to think with for non-technical educators and computer scientists). The fact that the *implementation model* is associated with an operational semantics can be used to simulate and/or run the script with an engine. As stated here, LD allows specifying some dimensions of script *implementation models*, but not in a straightforward and complete way. Differently, Miao & al. (2005) propose a language that allows a detailed specification of the script, mixing in an operational way what we have dissociated as the *structural* and *implementation models*. Based on the computer-science UML formalism (UML, 2006), this language is complete enough to model complex structures (e.g., sequencing activities with conditions and loops or managing complex distributions of roles), and is associated with authoring tools. With respect to SPAIRD, this corresponds to adopting a particular definition of the *structural model* (i.e., the script components) and of the type of *implementation model* that is targeted (i.e., the type of relations that can be defined between the components, the type of control structures that should be usable, etc.) and then, taking advantage of these choices, to provide the model with an operational semantics and some associated tools (e.g., authoring tool or language player). As another example of adopting a precise definition in order to target operational constructions, Haake & Pfister (2007) propose to describe scripts (roles, possible sequences of actions, etc.) as a finite state automaton, a formalism that allows complex control structures. The script can then be deployed on a platform that is compliant with this formalism. Within such an approach, the platform runs the script and provides access to functionalities or data according to the automata. The script can be modified at any time via its specification, without requiring any hand-modification of the platform, which provides a certain form of flexibility (within the principle of this approach, i.e., what can be modified is the program that specifies the way the system prompts students).

Finally, building script-specific platforms is a way to implement and articulate all aspects of the considered script. The obvious advantage is that it allows taking the different issues raised in the SPAIRD model into account, e.g., articulating script instructions and technological setting design decisions within specific tools (e.g., the Concept-Grid editor), realizing a specific integration of different functionalities within a single screen (e.g., the platform associated with RSC), managing script-specific data-flows and workflows, providing teachers with means to act dynamically on the setting, etc. The obvious drawback is that this approach requires a new platform to be constructed for every new script as a script-specific platform can be reused by changing the resources from one domain to another, but the principles are hard-coded. This drawback can however be limited by introducing tailorability means (cf. §5.3) and targeting platforms dedicated to a class of scripts. For instance, a platform could implement the Jigsaw family script's core mechanisms (e.g., controlling data-flow in order to expose different students to different knowledge, or controlling group composition on the basis of the conflicting criteria) whilst not fixing all the issues as a script-specific platform does. This approach allows implementing core aspects of the script whilst leaving some others open, and not having to construct a new platform for every new script. As an example, a platform that implements Concept-Grid principles whilst leaving different tuning options open to teachers is presented in Hong & Dillenbourg (2007).

In (Miao & al., 2005) the authors describe a list of support functions that can be obtained from modeling scripts. These support functions are related to the extent to which these models are understandable by a computer-based system, i.e., the extent to which fact they are described in a formal syntax and associated with an operational semantics. These support functions are: "system as editor/viewer" (using a modeling editor to build intermediation objects for the design team to think with and communicate on), "syntactical mapping to a visual/conceptual representation" (the model can be represented within a specific syntax, which allows checking it with respect to syntax correctness or data-exchanges with other software), "presentation of models in multiple perspectives" (the modeling dissociates different notions — e.g., temporal dimension or role dimension — and is precise enough to allow the system to present and analyze the script according to these points of view), "model-based prediction" (the system can advise the designers — e.g., highlight constrains or dependencies — with respect to some of the perspectives), "simulation" (the script can be simulated to identify possible issues such a deadlocks), "static configuration of the learning environment" (e.g., tuning a given platform from the script model), "monitoring the learning flow" (e.g., adapting the platform to the script enactment, sequencing the process or providing teachers or students with information on the process) and finally "model-based scaffolding" (e.g., advising learners on the best way to play their role). A conceptualization such as SPAIRD aims at the first level of support (intermediation/boundary objects for the design team to think with and communicate on). It however does not aim at addressing this level through a specific modeling language and associated model editor or viewer, but by just highlighting issues and interrelations, as we believe this intermediary level is also necessary. This is why we refer to it as a conceptualization model: this level allows not going in a too-straightforward way into a precise modeling language. The other side of the coin is that it provides only conceptual support. Conversely, precise modeling languages, such as the one presented in Miao & al. (2005), in particular when connected with visualization tools (Harrer & Malzahn, 2006), allow being more precise. They are more complex to use, require adopting the proposed modeling language notions and principles and going further into detailed design decisions then SPAIRD. The other side of the coin is that they propose in exchange more support (model editor, syntax checker) and (in this case) also advanced functions such as simulation. We see these different levels as complementary alternatives. With respect to Miao & al.'s support functions taxonomy, works such as Collage-Gricole, Bricolage or LAMS address as a first objective the static configuration of the learning environment objective (LAMS taking advantage of the fact it focuses on simple scripts and predefined embedded functionalities to also support some more advanced support functions).

5.2. Interests and difficulties of model-driven approaches for platform generation/tuning

In this section we will focus on the support function labeled as "configuration of the learning environment" in Miao & al. (2005). A common point of a conceptualization, such as SPAIRD, several aforementioned approaches to the operationalization of CSCL scripts (in particular, the platform generator and operationalization language approaches) and the current evolution of computer science is to highlight the interest of model-based approaches to support the building of technological settings, and in particular to generate automatically technological settings, which from the perspective of practice and dissemination is indeed an objective.

Considering platform generation, the major approach developed in software engineering is called Model Driven Architecture or MDA (MDA, 2003). This approach argues for the use of models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification of systems' computational implementation. MDA's principle is to use three abstraction levels and model transformation processes: (1) a computer-independent model (CIM) captures the environment and the general requirement for the system is built; (2) the CIM is transformed into a platform-independent model (PIM) that captures the specification issues that do not change from one technological platform to another; (3) finally, the PIM is transformed into a platform specific model (PSM) that combines the platform-independent issues with details related to the targeted platform. These transformations are based on machine-readable application and data models. They aim at automatically creating systems by code generation and/or software-component agglomeration (as opposed to handmade code), which is supposed to facilitate conceptual design, maintenance (including re-deploying the model on the latest hot technology), integration and testing.

At a general level, a conceptualization such as SPAIRD is a step forward towards a model-driven approach of script operationalization. As highlighted by the MDA vision, all computational systems (and we could add: all CSCL scripts) are based on some models, but most of these models remain immaterial in the head of the designers and are often created just before, and in the context of, a given operationalization. SPAIRD advocates and provides means to make explicit and use models to direct the course of understanding and design, and to facilitate the operationalization of scripts on current and future platforms. With respect to MDA, *pedagogical principles* and *intrinsic constraints* can be classified as computer-independent (CIM level). The *structural model*, *implementation model*, *platform specification* and *student oriented models* can be classified as platform-independent (PIM level) if described at an abstract level, i.e., if one refers to notions such as “group G1” and not to “Lucy and Bill”. More precisely, the *structural model* and *implementation model* can be seen as intermediate models that help in defining the *platform specification*, which is the transition model from the CIM level to the PSM level. *Extrinsic constraints* can be platform-independent or platform-dependent (e.g., when using a platform that imposes designers with constraints).

Several of the works we have analyzed here as possible particular and/or partial implementations of SPAIRD (as a general approach) go clearly into the direction advocated by MDA. Proposing patterns and authoring tools that generate an LD description which can then be deployed on an LD-compliant engine as proposed in the Collage-Gricode approach (Hernandez-Leo & al., 2007) is a straightforward step in this direction. Within this line of research, a core issue is that of what can be done in terms of automated transformations of an abstract model into an operational system. In Collage-Gricode, this is addressed at the level of orchestrating activities and providing access to the webservices that have been identified as functionalities useful for students, which is but an approach to integration. This can appear to be too limited if complex dynamic role distribution, data-accessibility flows or specific integration of functionalities within screens are required. The approach developed in Bricolage (Caron & al., 2005) is based on the use of the Meta Object Facility Specification (MOF, 2007), which allows computer-science models to be exported from one application, imported into another, transformed, and used to generate application code. Theoretically, this allows building a piece of software that (1) maps the two models corresponding to (i) a specification of the targeted technological setting (the platform specification) and (ii) an abstract model of a given piece of software (e.g., an LMS), and then to (2) automatically transform the code of this piece of software to make it comply with the platform specification, via a model-transformation process. This is however at present still a software engineering research issue.

SPAIRD as such can not be classified as a CIM. As highlighted previously, it is a descriptive and informative framework and is not associated with an operational semantics; it can thus not be used as such for automated transformations. Although fully convinced by the power of MDA-like approaches and automated model transformations, we believe such processes, by the fact they require precise modeling at early stages of the design of the script and platform, create a risk a too-quick/straightforward crystallization of these models. When the operationalization process is thought of as providing access to tools or functionalities within a basic common interface or with limited integration features (e.g., LD-like sequencing of activities), automatic transformation of a computer-independent model to a platform-specific model as proposed by the MDA process appears a promising approach. Automatic transformation may also appear powerful for addressing a limited specific concern (e.g., transformation of an interface or management of groups) for which a modeling language associated with an operational semantics can be identified as satisfactory. Such requirements remain in the scope of what can be, given the current technical state-of-the-art, managed automatically. Not misunderstanding this, our view is that macro-script operationalization should be generally addressed through manual (i.e., human-based) iterative transformations of models, within a process *informed* by a conceptualization such as SPAIRD and knowledge accumulated and refined through experience, and using modeling languages adapted to the considered setting when this becomes needed and the design is sufficiently advanced. Languages such as the ones proposed by Miao & al. (2005), Haake & Pfister (2007) or LD (IMS-LD, 2003) are not universal; they are based on, and carry, design decisions. Using at first a general conceptualization such as SPAIRD allows making explicit the issues and features to be considered, and the matters of concern. This is then a basis to orientate the implementation approach, and in particular the selection of a modeling language whose design options are in line with the considered matters of concern. Such a process greatly benefits from knowledge accumulated by experience and keeping track and questioning every design decision according to how the script has been enacted in different settings, including issues that are often neglected such as time issues (the first uses of a platform necessarily require an additional activity that corresponds to understanding how it works, and appropriating it to oneself). Manual transformations of models also allow taking advantage of the fact that, differently from most settings within which computer-based systems are constructed, a multidisciplinary team addressing the operationalization of a macro-script can act on both sides of the software: what the user is supposed to do (the script itself) and the computer-based system.

From a software engineering perspective, keeping coherence between the script and the platform is an important issue that can be related to the “structural correspondence principle” (Reinders & al., 1991) outlined by knowledge engineering researchers. This principle states that an explicit link between the conceptual design models and the platform characteristics (and vice versa) should be maintained as far as possible. In knowledge engineering, respecting such a structural correspondence has been demonstrated to be an important issue as (1) the multidisciplinary work that consists in working both on the models and the computer-based system is facilitated and (2) the understanding and the control of the computer-based system behavior is interpretable at an abstract level, using the model notions (and not only at the computer-science implementation language), which facilitates the tuning and the maintenance.

5.3. Technological-setting flexibility

We have emphasized that macro-script (and technological setting) perception and enactment are subject to unpredictable issues. This makes flexibility an important concern. Requests for flexibility may originate from the teachers and/or the students. For teachers, dealing with unpredicted events may require changing decisions (or making late decisions) according to the *actual interaction pattern*, or what can be perceived from some unattended *students' general perception*. For students, there may be a necessity to change some issues (e.g., groups or scheduling) or adapt the technological setting to needs in context, according to how the script is enacted and the underlying emergent issues, such as self-organization issues. Introducing flexibility in the technological setting is a means to address the fact that designing software to support activity is somewhat paradoxical as activity will emerge and is not fully predictable. Here again, this goes in the direction of the conclusions of Jones & al. (2006), that systems should be designed as "plastic forms that incline users to some uses in particular but are available to be taken up in a variety of ways."

Some requirements for script flexibility may be independent from any technological dimension. Some others, however, may be related to the technological setting. An obvious example is when the teacher or the students require adapting the accessible functionalities according to the emergence of new needs. Technological issues may also however arise in other cases, such as a modification of groups, which requires acting on the data-flow and managing data integrity (Dillenbourg & Tchounikine, 2007).

What comes with the notion of flexibility depends here again on the way the operationalization process and technological settings are thought of, and in particular the integration dimension. When the platform is thought of as providing self-service functionalities, flexibility is related to how new functionalities/tools can be added: by hand (e.g., by a teacher or a computer-scientist) or automatically (e.g., by updating the specification model and then its deployment). When the platform is thought of as an engine that runs an *implementation model* of the script (e.g., the approach proposed by Haake & Pfister (2007)), flexibility is introduced by the fact that this model can be changed at run time, e.g., skipping a phase or modifying groups, and then prompting students accordingly. When the platform is though of as a complex integration of interfaces, data flows and workflows, and some of these issues are to be modified run-time by teachers or students, then flexibility requires making the platform *tailorable*.

In computer science, a system is said to be *tailorable* if it provides its users with some means to modify itself in the context of its use as one of its functionalities (Malone & al., 1992; Morch, 1997). Tailorability is a means to combine the two objectives of (1) reifying the script's core mechanisms and (2) being flexible at run-time for both teachers and students. From another perspective, findings in knowledge engineering have demonstrated that *ad hoc* platforms have the apparent advantage of running the details of a model, but are in fact not as reusable as might be imagined because, in most cases, there is always a slight detail (typically related to the context of use) that differs from the prototype case implemented by the platform. This leads either to changing the model to fit the platform (which, in the case of scripts, would introduce a pedagogical bias), modifying the platform (which requires going into the code), or projecting the model on the platform (which leads to losing the structural principle). Tailorability is thus also a means to enhance platform reusability.

Introducing flexible/tailorable issues to CSCL platforms is an interesting research direction, but raises different issues. First, from a computer science point of view, tailorability is an issue. Platform generation and adaptation can be addressed by linking predefined software components with some software glue according to the script design (i.e., before running the script) and then during script enactment. Considering tools, this is easy when restraining the offer to a dedicated tool-repository, but is difficult if the objective is to allow run-time use of any tool students would like to use as it raises the problem of interoperating software components that have not been designed for this purpose. This is manageable when adopting the minimalist approach of allowing access to tools within a basic interface, but raises difficulties for script-specific interfaces or workflow issues. Moreover, tailorability must be technically easy as teachers and students can not be expected to be skilled programmers. This must thus be addressed with authoring tools (e.g., the Collage approach) or specific interfaces (e.g., the platform that supports student self-organization in RSC (Betbeder & Tchounikine, 2003)). Second, tailorability for students is to be studied with respect to the scope of flexibility defined by the intrinsic/extrinsic constraint notions, and potential teacher regulation. Third, tailorability is, with respect to the students' activity as related to the script, *another* activity; there is therefore a risk of causing a breakdown in the activity flow.

5.4. A high-level architecture of a model-based flexible script-engine

As a way to synthesize some of the major features we have discussed in this section and to present directions for future works, we propose in Figure 3 an abstract (theoretical) general picture of how a model-based flexible script-engine can be thought of. Our point here is not to advocate a "big-brother" engine, but rather to outline that a model-based approach allows considering different potentialities whose feasibility and educational interests are yet to be studied.

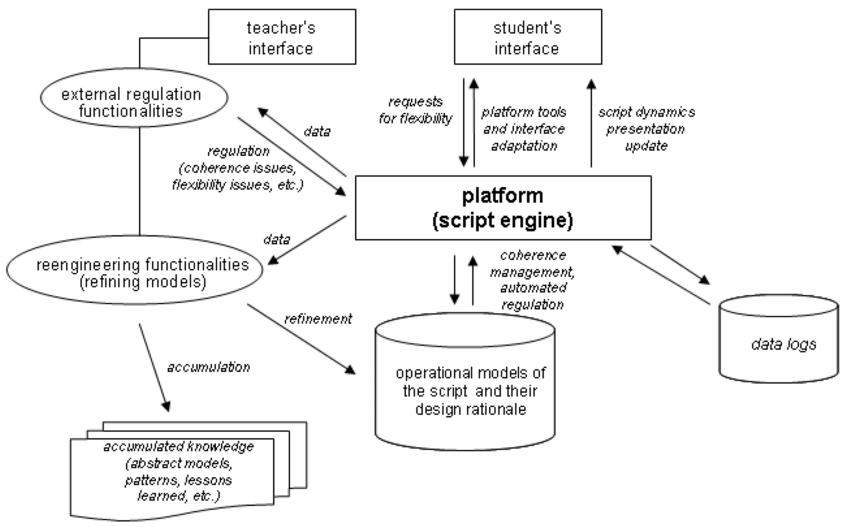
teacher's interface
student's interface
external regulation functionalities
data
regulation (coherence issues, flexibility issues, etc.)
requests for flexibility
platform tools and interface adaptation
script dynamics presentation update
platform (script engine)
data
reengineering functionalities (refining models)
coherence management, automated regulation
refinement
accumulation
operational models of the script and their design rationale
data logs
accumulated knowledge (abstract models, patterns, lessons learned, etc.)

Figure 3. A model-based flexible script-engine approach

Considering architectural design, a model-based script-engine must be based on an explicit representation of the script models and their design rationale. It can correspond to different types of software: hand-made script-specific platforms (e.g., a platform dedicated to the Concept-Grid script); generic platforms customized by hand according to the script (e.g., a particular instantiation of a CMS such as Zope/Plone); model-driven transformation of a framework (e.g., the Bricolage approach); engines associated with a more-or-less specific language (e.g., CopperCore and LD, works reported in Miao & al. (2005) or Haake & Pfister (2007)).

Considering capacities to run and manage the script, these abstract models must be associated with an operational semantics. This dimension is a key to different issues that arose in this article that may (according to the setting) be important matters of concern, such as:

- Orchestrate activities and manage the workflow, e.g., provide access to data or tools according to the *script structural and implementation models*. This is addressed by works such as Miao & al. (2005) and Haake & Pfister (2007), or in a more limited way when using LD.
- Adapt the *script dynamics presentation* (and, possibly, *the script and platform presentations*) in real time in order to maintain the coherence between the represented script and the *actual interaction pattern*.
- Manage the platform to comply with some students' or teachers' requests for flexibility whilst remaining coherent with the script's *intrinsic constraints*.
- Manage automatically some regulation issues and/or assist the teacher in his regulation (e.g., informing him of any unexpected event), here again with respect to the script's *intrinsic* and *extrinsic constraints*.

These issues fall into Miao & al.'s classifications of "monitoring of the learning flow," "dynamic configuration of the learning environment" and "model-based scaffolding" support functions.

Another important dimension of such architecture should be to support accumulation of knowledge by, for instance, supporting data analysis and recurrent-patterns identification, which will help in iteratively refining scripts, and in progressing in the understanding of script enactment.

Considering feasibility, the potential power of such a script-engine is correlated to the power of expression and precision of the used models (Miao & al., 2005). From the point of view of modeling, the issue is to elaborate languages that allow the teacher and the system to reason about the script issues, namely its design rationale, thus addressing computational issues through pedagogic notions (languages for teachers and languages for script-engines may be different but interoperable). From the point of view of the script-engine's potential, a first issue is the interpretation of the data and logs (computational events), a second issue is, if automated regulation is targeted, that of modeling-regulation decisions. Considering interpretation, some issues are intractable, for instance, regulations that require Natural-Language understanding can not be automated. In certain cases, artificially structured communication (i.e., using predefined sentences or sentence-openers) or computational-event analysis (e.g., tracking access to data or use of a function) can allow some kind of automated regulations and/or support teachers in their understanding of the students' actions. Some proposals already exist, such as UTL (Iksal & al., 2005), a dedicated language that aims at analyzing logs with respect to IMS-LD models. Referring to software engineering, things should probably be addressed by considering script classes/patterns and corresponding architectures, rather than "in general".

6. Conclusions

In this article we have proposed an analysis of some core issues that must be taken into account when operationalizing macro-scripts: a conceptualization model, an analysis of current technological approaches with respect to this model, and finally research directions for the design and implementation of technological settings that present the properties identified in our analysis. The presented model is proposed as an intermediation object for multidisciplinary work that can help in making clearer and elaborating knowledge of different issues, in particular: modeling of CSCL macro-scripts; understanding of the links scripts/technological settings (for both script and technological setting designers); design of technological settings; understanding of how technological settings can be analyzed in order to avoid being incoherent with the script and/or can be used to influence behavior in a way that is coherent with the script; studying the flexibility issues of scripts; understanding of script enactment; accumulating guidelines and knowledge that can be useful in orientating the selection of adequate technological settings, defining precise specifications for new technological settings, limiting the risks of technology-oriented choices that are in contradiction with pedagogic issues, or identifying what parameters can be tuned by the teachers and the students and what is not modifiable.

A conceptualization such as SPAIRD puts a set of issues on the worktable. An analogy can be made to software engineering approaches to development such as the object-oriented Unified Modeling Language (UML, 2006). First, different models denoting different dimensions are proposed and (if nothing else) designers then consider a set of issues just because of the fact these issues are outlined. Second, modeling languages are elaborated (or, as in the case of UML, already-existing modeling languages are reinterpreted in the context; at this level, we have emphasized that different existing languages could be linked with the view proposed by SPAIRD). Then, time and experience help in refining the overall conceptualization and the modeling languages, and in elaborating guidelines and knowledge. Finally, this can lead to the crystallization of an explicit process/methodology, similar to the software engineering “unified process” that is currently being elaborated at the top of UML. Within this perspective, SPAIRD is a contribution within the first phase. Questioning SPAIRD conceptualization both from a theoretical point of view and from practice, building on it or proposing alternatives will help in (1) developing intermediation conceptual tools for multidisciplinary research, (2) providing conceptual bases for design, (3) elaborating accumulated knowledge that can be used by designers to question their design and/or orientate decisions and possibly (4) identifying/building/reshaping modeling and operationalization languages within an articulated view.

From the perspective of accumulating knowledge, design-rationale approaches can be used in order to systematically identify, make explicit and keep track of design decisions (see Moran & Carroll (1996) for a reference book on design-rational approaches). For example, the QOC formalism (Question-Options-Criteria) helps in guiding and documenting a decision process by proposing a list of questions to be answered, together with possible options and selection criteria. Within this perspective, works such as SPAIRD or the conceptual model of scripts presented in Kobbe & al. (2007) help in making explicit and formulating questions related to script operationalization (e.g., “What are the options with respect to group formation, and the pertinent selection criteria?”). Premises for such a rationale approach to knowledge accumulation can be found in different research that intends to identify guidelines. For example, Strijbos & al. (2004) identified six steps for designing computer-supported group-based learning (defining the learning objective, the expected interaction, the task, the structure, the group size and how computer support can be best used) and, for every step, a set of key issues to be questioned. Dillenbourg & Jerman (2006) suggest building script repositories containing abstract models of scripts, structured and indexed according to different issues such as pedagogic principles, script families or structural or implementation characteristics. Other works attempt to accumulate and share knowledge as patterns, providing teachers and designers with comprehensive and structured design ideas, both ready to use (and/or adapt) and coupled with research evidence (Goodyear, 2004; DiGiano & al., 2002; Hernandez Leo & al., 2004). Such a perspective can be addressed by both theoretical means and knowledge accumulation means. Research in knowledge engineering such as the KADS methodology (Wielinga & al., 1992) have powerfully demonstrated how libraries of generic models (in KADS, problem-solving models) could be constructed by mixing reverse engineering (i.e., analyzing *a posteriori* existing systems), abstraction and generalization processes, and how such libraries are powerful help for system designers. Such works are not only useful for disseminating scripts and making them accessible to practitioners, they also raise interesting research issues with respect to the levels of abstraction and genericity that should be used, what can be addressed at the different levels (e.g., generic level, script-family level, script abstract level or specific-setting level), and how technological settings can be associated with such abstract models.

7. Acknowledgements

This work benefited from fruitful exchanges and collaboration with P. Dillenbourg (Craft, EPFL, Switzerland). The author thanks C. Choquet (Lium, University of Le Mans, France), A. Harrer (Collide, University of Duisburg-Essen, Germany), C. Jones (OpenUniversity, UK), L. Kobbe (KMRC, Tübingen, Germany), and the anonymous reviewers for their comments on preceding versions of this article. More generally, this research benefited from the work realized in the groups focusing on CSCL of Kaleidoscope (http://www.noe-kaleidoscope.org), a European Research Network of Excellence focusing on Technology Enhanced Learning.

8. References

Baker, M.J., Lund, K. (1997). Promoting reflective interactions in a computer-supported collaborative learning environment. Journal of Computer Assisted Learning, 13, 175-193.

Brousseau, G. (1998). Théorie des situations didactiques. La Pensée Sauvage (Ed), France.

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: European Conference on Computer Supported Collaborative Learning (pp. 97-105), Maastricht (NL).

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

Bote-Lorenzo, M.L., Hernández-Leo, D., Dimitriadis, Y., Asensio-Pérez, J.I., Gómez-Sánchez, E., Vega-Gorgojo, G., Vaquero-González, L.M. (2004). Towards Reusability and Tailorability in Collaborative Learning Systems Using IMS-LD and Grid Services. Advanced Technology for Learning, 1(3), 129-138.

Caron, P-A, Derycke, A, Le Pallec, X. (2005). Bricolage and Model Driven Approach to design distant course. In proceedings of the world conference on E-learning in corporate Government, Healthcare & higher education (pp 2856-2864), Vancouver (Canada).

CopperCore (2007). http://coppercore.sourceforge.net/index.shtml. Last visited March 2007.

Delium, C. (2003). OSCAR: a framework for structuring mediated communication by speech acts. In: IEEE International Conference on Advanced Learning Technologies (pp. 229-233), Athens (Greece).

DiGiano, C., Yarnall, L., Patton, C., Roschelle, J., Tatar, D. G., Manley, M. (2002). Collaboration design patterns: Conceptual tools for planning for the wireless classroom. In: International Workshop on Wireless and Mobile Technologies in Education (pp. 39-47), Växjö (Sweden).

Dillenbourg, P., Jermann, P. (2006). 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. In press.

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

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.

Dillenbourg, P. (1999). What do you mean by collaborative learning? In: P. Dillenbourg (Ed) Collaborative-learning, Cognitive and Computational Approaches (pp. 1-19). Oxford: Elsevier.

Engeström, Y. (1987). Learning by expanding. A activity-theoretical approach to developmental research. Helsinki: Orienta-Konsultit.

Ferraris, C., Martel, C., Vignollet, L. (2007). LDL for Collaborative Activities. In: Botturi, L., Stubbs, T. (eds) Handbook of Visual Languages in Instructional Design: Theories and Practices. Hershey, PA: Idea Group (in press)

Fischer, F., Mandl, H., Haake, J., Kollar, I. (2007). Scripting Computer-Supported Collaborative Learning – Cognitive, Computational, and Educational Perspectives. Computer-Supported Collaborative Learning Series, New York: Springer.

Goodyear, P. (2004). Patterns, pattern languages and educational design. In R. Atkinson, C. McBeath, D. Jonas-Dwyer & R. Phillips (Eds), Beyond the comfort zone: ASCILITE Conference (pp. 339-347), Perth (Australia).

Goodyear, P. (2001). Effective networked learning in higher education: notes and guidelines. Volume 3 of the Final Report to JCALT: Networked Learning in Higher Education Project. Retrieved from http://csalt.lancs.ac.uk/jisc/ (December 2nd, 2006).

Haake, J., Pfister, H.-R. (2007). Flexible scripting in net-based learning groups. In F. Fischer, I. Kollar, H. Mandl & J.M. Haake (Eds.), Scripting computer-supported cooperative learning. Cognitive, computational, and educational perspectives. New York: Springer.

Harrer A., Malzahn, N. (2006). Bridging the gap –towards a graphical modelling language for learning designs and collaboration scripts of various granularities, In: International Conference on Advanced Learning Technologies (pp. 296-300), Kerkrade (The Netherlands).

Hernández-Leo, D., Burgos, D., Tattersall, C., & Koper, R. (2007). Representing CSCL macro-scripts using IMS LD: lessons learned. Submitted. URI: http://hdl.handle.net/1820/784.

Hernández-Leo, D., Villasclaras-Fernández, E.D., Jorrín-Abellán, I.M., Asensio-Pérez, J.I., Dimitriadis, Y., Ruiz-Requies, I., Rubia-Avi, B. (2006). Collage, a Collaborative Learning Design Editor Based on Patterns Special Issue on Learning Design. Educational Technology & Society 9(1), 58-71.

Hernández-Leo, D., Asensio-Pérez, J.I., Dimitriadis, Y., Bote-Lorenzo, M.L., Jorrín-Abellán, I.M., Villasclaras-Fernández, E.D (2005). Reusing IMS-LD Formalized Best Practices in Collaborative Learning Structuring. Advanced Technology for Learning 2(3), 223-232.

Hernández-Leo, D., Asensio-Pérez, J.I., Dimitriadis, Y. (2004). IMS Learning Design Support for the Formalization of Collaborative Learning Patterns. In: International Conference on Advanced Learning Technologies (pp. 350-354), Joensuu (Finland).

Hong, F., Dillenbourg, P. (2007). Conveying a pedagogical model through design patterns. In: Workshop on Computer-Supported Collaboration Scripts, Villars (Switzerland). Retrieved March 2nd, 2007 from http: http://www.iwm-kmrc.de/cossicle/workshop/resources.html.

Iksal, S., Choquet, C. (2005). Usage Analysis Driven by Models in a Pedagogical Context. In: AIED'05 Workshop on Usage analysis in learning systems (pp. 49-56), Amsterdam (NL).

IMS-LD (2003). IMS Global Learning Consortium. IMS Learning Design v1.0 Final Specification. Retrieved May 29th 2006, from http://www.imsglobal.org/learningdesign/index.cfm.

Jermann, P., Dillenbourg, P. (2003). Elaborating new arguments through a CSCL scenario. In: Arguing to Learn: Confronting Cognitions in Computer-Supported Collaborative Learning Environments (pp. 205-226). Kluwer, Amsterdam.

Jones, C., Dirckinck-Holmfeld L. & Lindström, B. (2006) A relational, indirect, meso-level approach to CSCL design in the next decade. IJCSCL 1(1), 35-56.

Kirschner, P., Strijbos, J.W., Kreijns, K., Beers, P.J. (2004). Designing Electronic Collaborative Learning Environments. Education Technology Research & Development 52(3), 47-66.

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

Le Moigne, J.-L. (1990). La modélisation des systèmes complexes. Paris, Dunod.

LAMS (2007). http://www.lamsinternational.com. Last visited March 2007.

Malone, T. W., Lai, K.-Y., Fry, C. (1992). Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. In: ACM Conference on Computer-Supported Cooperative (pp 289-297), Toronto (Canada).

MDA (2003). OMG, MDA guide version 1.0.1, Retrieved May 29th 2006, from http://www.omg.org/mda/.

McGrenere, J., Ho, W. (2000). Affordances: Clarifying and Evolving a Concept. In: Proceedings of Graphic Interface (pp. 179-186), Montreal (Canada).

Miao, Y., Hoeksema, K., Hoppe, U., Harrer, A. (2005). CSCL scripts: modelling features and potential use. In International Computer Supported Collaborative Learning Conference (CD-Rom), Taipei (Taiwan).

MOF (2007). Meta-Object Facility http://www.omg.org/mof/. Last visited March 2007.

Moran, T. P., Carroll, J.M. (1996). Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum Associates.

Morch, A. (1997). Three Levels of End-user Tailoring: Customization, Integration, and Extension. In: M. Kyng and L. Mathiassen (eds), Computers and Design in Context (pp 51-76), The MIT Press, Cambridge MA.

Norman, D.A. (1999). Affordances, conventions, and design. Interactions 6 (3), 38-43.

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: Lawrence Erlbaum Associates.

Pinkwart, N. (2003). A Plug-In Architecture for Graph Based Collaborative Modeling Systems. In U. Hoppe, F. Verdejo, & J. Kay (Eds.): Shaping the Future of Learning through Intelligent Technologies, proceedings of the 11th Conference on Artificial Intelligence in Education (pp. 535-536). Amsterdam (NL).

Plone. Plone: A user-friendly and powerful open source Content Management System. Retrieved May 29th 2006, from http://www.plone.org.

Rabardel, P. (2003). From artefact to instrument. Interacting with Computers 15(5), 641-645.

Reinders, M., Vinkhuyzen, E., Voss, A., Akkermans, H., Balder, J., Bartsch-Spörl, B., Bredeweg, B., Drouven, U., van Harmelen, F., Karbach, W., Karsen, Z., Schreiber, G., Wielinga, B. (1991). A conceptual modelling framework for knowledge-level Reflection. AI Communications 4(2/3), 74-87.

Schmidt, K. (1990). Analysis of Cooperative Work. A Conceptual Framework. Roskilde, Denmark, Riso National Laboratory.

Soller, A. (2001). Supporting social interaction in a collaborative learning system. International Journal of Artificial Intelligence in Education 12(4), 40-62.

Strijbos, J.W., Martens, R.L., Jochems, W.M.G. (2004). Designing for interaction: Six steps to designing computer-supported group-based learning. Computers in Education 42, 403-424.

Suthers D., Weiner A. (1995). Groupware for Developing Critical Discussion Skills. In John L. Schnase and Edward L. Cunnius, Eds. Proceedings of CSCL '95, the First International Conference on Computer Support for Collaborative

Learning. Oct. 17-20, 1995. Indiana U. Bloomington, IN. Mahwah, NJ: Lawrence Erlbaum Associates (pp. 341-348). See also http://belvedere.sourceforge.net.

Tchounikine P. (2007). Directions to acknowledge learners' self organization in CSCL macro-scripts. In: Haake J.M., Ochoa S.F., Cechich A. (eds) Groupware: Design, Implementation and Use, LNCS 4715 (pp 247-254), Springer Berlin / Heidelberg.

UML. Unified Modeling Language, OMG, Retrieved May 29th 2006, from http://www.uml.org.

Wasson, B., Morch, A. (2000). Identifying Collaboration Patterns in Collaborative Telelearning Scenarios. Educational Technology & Society, 3(3), 237-248.

Wielinga, B., Schreiber, A., Breuker, A. (1992), Kads: a modelling approach to knowledge engineering. Knowledge Acquisition journal 4(1), 1-162.

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

Zope. Zope community, Retrieved May 29th 2006, from http://www.zope.org.