The mechanics of CSCL macro scripts

Pierre Dillenbourg, Fabrice Hong

Computer-Supported Collaborative Learning (2008) 3:5–23
DOI 10.1007/s11412-007-9033-1

The mechanics of CSCL macro scripts

Pierre Dillenbourg · Fabrice Hong

Received: 12 June 2006 / Accepted: 9 November 2007 /
Published online: 8 January 2008

Abstract Macro scripts structure collaborative learning and foster the emergence of knowledge-productive interactions such as argumentation, explanations and mutual regulation. We propose a pedagogical model for the designing of scripts and illustrate this model using three scripts. In brief, a script disturbs the natural convergence of a team and in doing so increases the intensity of interaction required between team members for the completion of their collaborative task. The nature of the perturbation determines the types of interactions that are necessary for overcoming it: for instance, if a script provides students with conflicting evidence, more argumentation is required before students can reach an agreement. Tools for authoring scripts manipulate abstract representations of the script components and the mechanisms that relate components to one another. These mechanisms are encompassed in the transformation of data structures (social structure, resources structure and products structure) between script phases. We describe how this pedagogical design model is translated into computational structures in three illustrated scripts.

Keywords Scripts · Pedagogical design model

Introduction

When two students work together, one often focuses on low-level aspects of the task while the other spontaneously pays more attention to metacognitive aspects (Miyake 1986; O'Malley 1987). As collaboration proceeds, these roles may shift, with the regulator becoming the regulated and vice-versa. Under certain conditions, this mutual regulation process is internalized by the students, giving rise to the development of self-regulation skills (Blaye et al. 1991). This phenomenon has been observed by psychologists and can be converted into a pedagogical method by explicitly assigning each of the two roles to student

P. Dillenbourg (✉) · F. Hong
School of Computer and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
e-mail: pierre.dillenbourg@epfl.ch

F. Hong
e-mail: Fabrice.hong@epfl.ch

peers. In the reciprocal tutoring method, for instance, one student reads a paragraph and the second asks comprehension-monitoring questions before roles are switched for the next paragraph (Palincsar and Brown 1984). This turn-taking mechanism could further be translated into a physical artifact. Imagine a table in the form of a see-saw commonly found in children's playgrounds (Fig. 1). When the student who is at the lower end of the see-saw is able to manipulate the objects on the table, the second student is automatically at the higher end and has a more global view. This table mechanism translates the pedagogical design into a physical event: if one student is carrying out the task, the other is automatically forced to adopt a regulatory role.

This example is expandable to other pedagogical methods. For instance, socio-cognitive conflict can be triggered by placing students at two orthogonal viewpoints of a pyramid (Doise and Mugny 1984). These psychological studies lead to pedagogical methods which trigger epistemic conflicts among team members. This mechanism can also be adapted to the table example: The solid axis between chairs could be formed in such a way as to force students to adopt perpendicular viewpoints.

The pedagogical methods addressed in the following sections rely on mechanisms that induce specific interactions in the same way that the mechanics of the two table examples would influence interactions. While pedagogical mechanisms are usually described by verbal principles, they can be translated into physical mechanisms (as in the table examples above) and of course into computational mechanisms. The purpose of the present paper is to describe these rather simple computational mechanisms.

Why scripting?

Collaborative learning is not always effective; its effects depend on the richness and intensity of interactions engaged in by group members during collaboration (Dillenbourg et al. 1996). Learning outcomes are related to the emergence of elaborated explanations, the negotiation of meanings, the quality of argumentation structures and the mutual regulation of cognitive processes. The holy quest of CSCL is to establish environments that directly or indirectly favor the emergence of rich interactions. This is commonly referred to as 'design for conversations' (Roschelle 1990) or 'designing interactions' (Jermann et al. 2001; Dillenbourg and Fischer 2007). Many non-exclusive approaches exist by means of which a CSCL environment can directly or indirectly shape group interactions, namely:

1. By designing a communication tool, for instance semi-structured interfaces, that proposes predefined speech acts in the form of buttons or sentence openers (Baker and Lund 1996; Veerman and Treasure-Jones 1999; Soller 2001);
2. By shaping (graphical) representations of the task and the objects to be manipulated by students (Roschelle 1990; Suthers 1999);

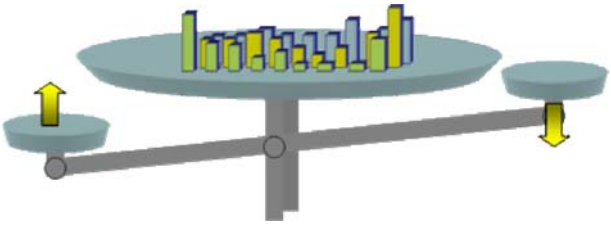

Fig. 1 A pedagogical method translated into a physical mechanism

3. By forming groups in a specific way (Wasson 1998; Hoppe and Ploetzner 1999; Inaba et al. 2000; Muehlenbrock 2006; Wessner and Pfister 2001)
4. By providing team members with a representation of their interactions in order to foster regulation at the group level (Dillenbourg et al. 2002; Jermann and Dillenbourg 2007)
5. By providing feedback on the quality of group interactions (McManus and Aiken 1995; Inaba and Okamoto 1996; Ayala and Yano 1998; Barros and Verdejo 2000; Constantino-Gonzalez and Suthers 2000). See also various similar approaches as reviewed by Jermann et al. (2001)
6. By scripting the collaboration process using specific phases, roles and activities.

The term 'script' has been used to refer to two approaches that share the same goal but are nonetheless different:

– Micro-scripts are dialogue models, mostly argumentation models, which are embedded in the environment (Approach 1 above) and which students are expected to adopt and progressively internalize. For instance, a micro-script may prompt a student to respond to the argument of a fellow student with a counter-argument (Weinberger et al. 2002).
– Macro-scripts are pedagogical models, i.e. they model a sequence of activities (a combination of Approaches 3 and 6 above) to be performed by groups. For instance, argumentation can be triggered by collecting students' opinions and pairing students with conflicting opinions.

The terms micro/macro refer to the granularity of the prescribed actions: a four-second turn of dialogue in micro-scripts versus a 4-h task in macro-scripts. A more important difference between micro and macro scripts is the fact that, in contrast to pedagogical models (macro scripts), dialogue models (micro scripts) are expected to be internalized. The present paper focuses on macro-scripts.

Examples of CSCL macro-scripts

Three examples of macro scripts developed by the present authors will be presented in the following. While the first two scripts (ArgueGraph and ConceptGrid) have been multiply tested and are integrated in a script authoring environment (http://manyscripts.epfl.ch), the third script (WiSim) has not yet been tested in a real context.

ArgueGraph

The aim of ArgueGraph (Jermann et al. 1999) is to trigger argumentation between peers. This is achieved by collecting opinions and forming pairs of students with opposite opinions. An experiment showed that conflicts led pairs to produce justifications that were not proposed by any peer in the individual phase (Jermann and Dillenbourg 2003).

Phase	Level	Activity
1	Individual	Each student responds to an on-line multiple choice questionnaire. These questions do not have right or wrong answers, but rather reflect different viewpoints. These viewpoints are to be addressed by the teacher in Phase 4. For each answer, the student is expected to write a few lines justifying his or her choice.
2	Class	Based on the responses in Phase 1, the system produces a map of opinions (see Fig. 2 / left). The teacher discusses this map with students and forms pairs in such a way as to

		maximize distance between students, i.e. pairing of students who provided conflicting responses in Phase 1.
3	Group	Pairs answer the same questionnaire as in Phase 1. The environment provides them with the answers and justifications provided by each peer in Phase 1. Pairs must select a single answer.
4	Class	This debriefing session aims to reformulate the elements mentioned by the students using the correct terminology, to structure them and to integrate them into a theoretical framework. The teacher synthesizes the arguments provided by the students (Fig. 2, right), asks them to provide further clarification, rephrase their justifications, compare them and so forth.
5	Individual	Each student chooses one of the questions discussed and writes a summary of all arguments collected by the system in Phases 1 and 3, structured according to the framework developed in Phase 4.

Similar approaches have been developed by pairing students with different hypotheses in an inquiry-based learning environment (Gijlers and de Jong 2005) (Fig. 2).

ConceptGrid

ConceptGrid is a sub-class of the JIGSAW scripts (Aronson et al. 1978). Team members acquire complementary knowledge by reading different papers. The concept grid can only be constructed when each member explains the concepts about which he/she has individually read (Fig. 3).

Phase	Level	Activity
1	Group	The group distributes the roles among its members. Each role is associated with a few papers to read.
2	Individual	Each student reads the papers associated with his or her role.
3	Group	The group distributes the concepts to be defined among its members.
4	Individual	Each student enters a 5–10-line definition of the concepts (s)he has been assigned.
5	Group	The group constructs a concept grid, i.e. concepts are ordered on a map in such a way that two neighbouring concepts can be explained in just a few sentences (Fig. 3).
6	Class	This debriefing session aims to reformulate the definitions and relations provided by the students, to structure them and to integrate them into a theoretical framework.

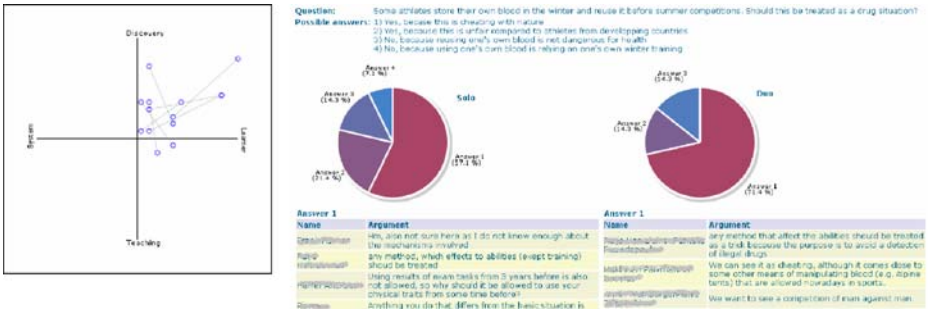

Fig. 2 ArgueGraph: *Left* the opinions map in Phase 2; *Right* the tool collects all answers for debriefing (Phase 4)

Fig. 3 Students are required to construct a grid in which they explain the relationship between juxtaposed concepts

WiSim

WiSim addresses two pitfalls of enquiry-based learning (De Jong and Van Joolingen 1998): the lack of explicit negotiation of parameters (Phase 3) and the difficulty of interpreting the representations of results (Phase 6). Script software is run on mobile phones and exploits the fact that mobile devices are often exchanged among teenagers. WiSim is designed for co-presence situations: co-located groups of students interact face-to-face and use their mobile phone as tiny computers to support their activities.

Phase	Level	Activity
1	Class	The teacher conducts an introductory lecture providing background knowledge on the phenomenon to be simulated.
2	Group	Students physically form groups in the class. One group member creates the group in the database via his/her phone application and the other members join by entering the name of the group to which they belong on their phone.
3	Group	Team members negotiate the values of the parameters which are to be entered by each member.
4	Individual	Each team member sends his or her parameter values to the simulation using his/her mobile phone (Fig. 4).
5	Individual	Each student receives a different representation of the same simulation results.
6	Group	The group compares the different representations of the results they have obtained.
The group repeats Phases 3 to 6 several times.		
7	Class	This debriefing session aims to synthesize the results of the simulation, by for instance, comparing all collected values on a graph and integrating them into a theoretical framework.

Script concepts

Despite the first “C” in CSCL, the scripts presented above are not restricted to computer-based activities, but integrate a variety of mediated or face-to-face activities across different spaces. The scripts are predominantly used in co-present settings and aim to enrich traditional university teaching through collaborative learning activities (Fig. 4).

Despite the second “C” in CSCL, these scripts are not purely ‘collaborative’, but include both individual activities such as reading a paper or writing a summary, and collective

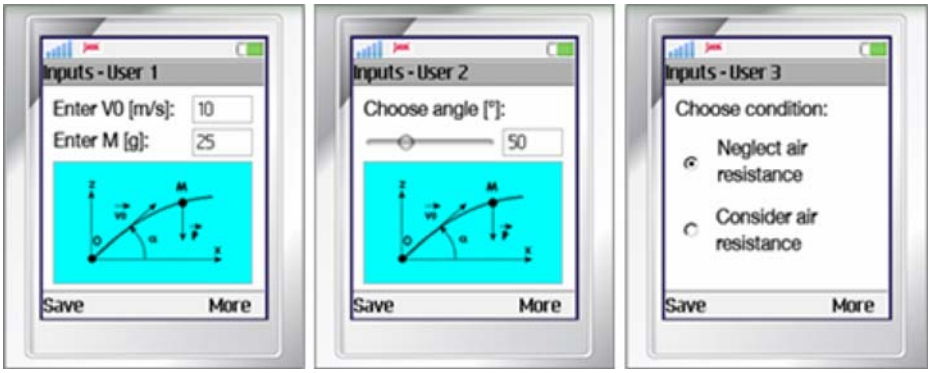

Fig. 4 Each team member sets up different simulation parameters in WiSim

(class-wide) activities such as debriefing sessions. We do not see any reason to develop pedagogical methods which exclusively rely on group activities. Individual reflection is required in order to transform experience into learning and class-wide activities are especially valuable when it comes to structuring the informal knowledge that emerged in previous phases. Figure 5 depicts the distribution of activities across these three social planes for the script instances presented above.

In other words, the scripts presented are neither purely computerized, nor purely collaborative. We termed this two-fold extension 'integrated scripts'; the scripts integrate several activities distributed across multiple places and different social planes (individual, collaborative, collective) within a single workflow. This broadened script concept places the teacher back at the centre of activity, not for lecturing, but rather for "orchestrating" (Randolph and Evertson 1994; Dillenbourg and Fischer 2007) script activities. Hence, our script environment includes features that enable the teacher to follow student activities in real time and, if necessary, modify the script, for instance by changing groups or deadlines (Dillenbourg and Tchounikine 2007).

Authoring scripts

The similarities among the script examples presented so far and the scripts developed by our colleagues (see Acknowledgments section) raise the expectations that a large number of scripts could be produced simply by combining basic script components. This led a group of European scholars to describe scripts using a set of components and operators (Kobbe et al. 2007). Components describe the sequence of activities in an abstract way, as in any authoring language or educational meta-data. A number of script authoring tools have been developed over the last few years and have encountered growing interest, e.g.

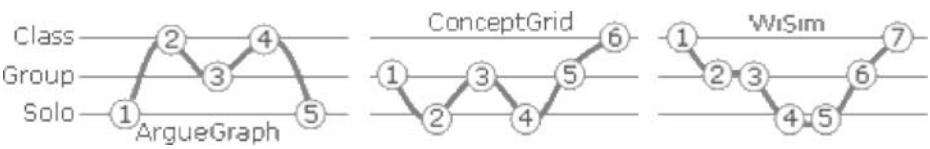

Fig. 5 Distribution of script activities across three social planes (numbers refer to phases)

CeLs (Ronen et al. 2006) and LAMS (http://www.lamsinternational.com). The ArgueGraph and ConceptGrid scripts are embedded in an authoring environment that enables the designer to edit the contents of the script and to tune certain parameters such as group size (in ConceptGrid). This environment (manyscripts.epfl.ch) differs from that of CeLs or LAMS: it has a low level of abstraction, only enabling the editing of scripts that are predefined, but it implements the scripting mechanisms addressed in this paper. ManyScripts handles three levels of abstraction (Dillenbourg and Jermann 2007), presented here in descending order:

- *Scripts*. ManyScripts stores scripts such as ArgueGraph or ConceptGrid independently of content.
- *Script instances*. The designer may select a script and provide it with specific content (questions in ArgueGraph, roles, concepts and readings in ConceptGrid), producing an instance, such as "ConceptGrid-on-Knowledge-Management".
- *Script sessions*. To run a script instance with a specific class of students, the teacher creates a script session. This session holds data such as students' names and the timing of each phase. Initial session data are supplemented by runtime data such as, in the case of the ConceptGrid, the composition of the groups, the definitions produced by the students and the position of the concepts on the grids.

A possible higher level of abstraction, script families, discriminates classes of scripts which use the same pattern: ConceptGrid uses the Jigsaw pattern i.e. it distributes knowledge among group members; ArgueGraph relies on a conflict-raising pattern; the Reciprocal Tutoring script (Palincsar and Brown 1984) relies on a mutual regulation pattern. This level of abstraction is not represented in ManyScripts.

The efficiency of a script is not intrinsic to the script definition, but also depends on the specific instance and the session. For example, the potential learning outcomes of ArgueGraph not only depend on the quality of the scenario per se, but also upon the quality of the questions (do they lead students to express divergent or convergent answers?) as well as on the extent to which the students engaged in a specific session have different *ab initio* opinions on the topics addressed by the questions.

This paper does not aim to present script authoring tools per se, but to address the scientific challenge underlying the development of such an environment. One frequently addressed challenge is the elaboration of ontologies describing the learning activities that are to be edited and sequenced by authors. We explore a second challenge, namely, how to capture the dynamics of a script, i.e. the mechanics referred to in the introduction. The mechanisms that favor the emergence of productive interactions lie in the relationship between activities and are located in transformations of the underlying data structures. While addressing data structures may sound like a back-office technological detail, we actually consider the key differences between a CSCL script and a genuine lesson plan to be embedded in the underlying workflow and data structures. In addressing the challenge of capturing script mechanics, we use a model that articulates these data mechanisms with the raison d'être of a script, i.e. scaffolding targeted types of interaction.

SWISH mechanisms

The SWISH model relates emerging group interactions to the way in which the task or its resources are distributed across team members. It may appear counter-intuitive to formalize

task distribution given that collaborative learning is often defined as the process of constructing and maintaining a shared understanding of the task (Roschelle and Teasley 1995). However, the effects of group learning are not dependent on the construction of a shared understanding per se, but rather on the effort exerted by group members to develop a shared understanding despite their differences (Schwartz 1995). Macro scripts engineer these differences: a script tunes the effort which is necessary for students to reach a shared understanding. Effort refers to the frequency and quality of the previously mentioned types of interaction: explanations, argumentation, and mutual regulation. The methods adopted in tuning the collaborative effort vary across the three presented scripts:

- ArgueGraph increases collaborative effort by selecting conflicting pairs and hence augmenting the argumentation which is necessary to reach an agreement.
- ConceptGrid increases collaborative effort by fragmenting knowledge (readings) and hence augmenting the need for explanation among team members.
- WiSim increases collaborative effort by distributing simulation inputs across different phones and hence requiring students to negotiate values and coordinate their experimental design.

For the tuning of collaborative effort, each of the scripts creates some kind of difference or 'split' among the students involved:

- ArgueGraph relies on the differences of opinion that naturally exist among students in order to form pairs.
- ConceptGrid creates artificial differences in the knowledge among peers by providing them with different resources to read.
- WiSim creates role differences by allocating different responsibilities in setting up parameters.

In these three examples, script designs involve disturbing a 'natural' collaborative system in such a way that the interactions necessary to maintain/restore collaboration produce the desired learning outcomes. We refer to this design principle as "Split Where Interaction Should Happen" (SWISH). It can be summarized in three axioms:

1. Learning results from the interactions in which students have to engage in order to compensate for the 'split' introduced by the script, i.e. constructing a shared solution despite distributed resources, knowledge, etc.
2. The nature of the 'split' thus determines the nature of interactions. Interactions are mechanisms for overcoming task splits.
3. Task splits can therefore—in reverse engineering—be designed to trigger the very interactions that the designer wants to foster: Split Where Interaction Should Happen.

Increasing collaborative effort too much may of course also damage collaboration (Dillenbourg 2002). As is often the case in educational design, a trade-off is required: while psycholinguists describe the mechanisms by which natural conversation tends to minimize collaborative effort (Clark and Wilkes-Gibbs 1986), collaboration breaks down and effects such as the free rider effect emerge if understanding one another becomes too difficult. The aim of educational design is to achieve the best compromise between ease and workload, a trade-off that we term 'optimal collaborative effort' (Dillenbourg and Traum 2006). The same trade-off is also targeted in discovery learning methods, as expressed in the term 'guided discovery'.

Data structures

SWISH is not a computational but rather a pedagogical design model, i.e. a way for designers to think about scripts. The essence of SWISH is not embedded in the description of a sequence of activities but encrypted in the data structures that underlie the script execution. To develop a computational account of SWISH, it is thus necessary to relate its principles to data structures and operators. We describe data structures at an abstract level, i.e. independent of their implementation. At this level, the mechanics of scripts can be described using simple tables or matrices and with a limited set of operators.

Social structure

Macro script activities occur across three social planes (see "Script concepts" section): the whole class, a student group and the individual student. Both the class and the set of individuals remain static throughout the script, whereas groups are dynamic structures that may evolve with time, e.g. when forming groups dynamically on the basis of individual behavior or on the basis of rotation. In many scripts, each group member is required to play a specific role. These four basic elements (students, groups, class and roles) define the social structure of any script. This simple structure enables the definition of phase transitions as matrix transformations based on generic operators.

Some scripts do not differentiate roles within teams. This situation can be simply represented by inserting the same role into each row of the matrix. Roles and groups do not always define a strict partition of a class. The social structure does not presuppose that a student belongs to only one cell or that one cell contains a single student (a matrix may have two students per role or two roles per student). Such cases require, however, special attention when it comes to implementation. Tolerating 'imperfect' social structures is necessary in order to guarantee flexibility of CSCL scripts. This includes, for example, coping with student number 17 when groups of four are to be formed in a class of 17 students, continuing with the script when one team member drops out of the class, etc. Two ways of providing flexibility have been implemented in the ConceptGrid environment: the "joker" and "spy" roles (Dillenbourg and Tchounikine 2007). The 'joker' is an extraneous group member who is allowed to play any role within the group (see Armin in Group3, Table 1). The 'spy' role allows members of the incomplete Group2 (Table 1) to borrow products produced by those members of another group who play the role that is missing in Group2.

In some scripts, the social structure changes over time. Some JIGSAW scripts include phases in which students with $role_x$ in their respective team meet with those playing the same role in other teams. These two types of groups (groups comprising members with different roles and groups comprising members with identical roles) are respectively represented on the left and right hand sides of Table 2.

Table 1 Social structure of CSCL scripts

Class	$Group_1$	$Group_2$	$Group_3$	….	$Group_i$
$Role_1$	Pierre	Michel	Jacques		
$Role_2$	Jean	Emma	Lena		
$Role_3$	Sophie	<SPY>	Tamara		
<Joker>			Armin		

Table 2 Social matrix rotation in a Jigsaw script

Class	$Group_{H1}$	$Group_{H2}$	$Group_{H3}$	Class	$Group_{E1}$	$Group_{E2}$	$Group_{E3}$
$Role_1$	Amy	Andrea	Alberto	$Role_0$	Amy	Bernie	Marie
$Role_2$	Bernie	Brigitta	Bruno	$Role_0$	Andrea	Brigitta	Michela
$Role_3$	Marie	Michela	Marco	$Role_0$	Alberto	Bruno	Marco

Such a rotation was embedded in the UniverSanté script (Berger et al. 2001), which combined co-present and distance argumentation phases. The script was used by medical students from Switzerland, Lebanon and Tunisia for a course on public health. In co-present meetings, students from the same country studied different diseases (Table 3, left) while, in distance meetings, students from different countries studied the same disease (Table 3, right).

Roles and resources

Roles are key components of script mechanics. The notion of "role" is a synthetic way to differentiate the contributions expected from team members and to communicate these expectations to the students. For instance, one team member may be asked to adopt the role of team coordinator in a project-based script or to play 'Piaget' in a pedagogical debate. A role may simply be a label from which students can infer what they are supposed to do. Designating roles is one way of defining the 'didactic contract' (Brousseau 1998). We encounter two types of roles in CSCL scripts, 'induced' roles and 'natural' roles. Induced roles are roles proposed by the script with a low or high degree of coercion. Natural roles refer to the exploitation of differences that exist between students before entering the script.

Roles can be induced by restricting access to specific resources, documents or tools, or by defining specific responsibilities related to student authentication in the system.

- *Documents*. For instance, in the ConceptGrid, a student adopts the role of Jean Piaget by reading three papers written by Piaget; papers that the other team members will (probably) not read. While the documents associated with a particular role are accessible to other roles, students tend to focus on those resources for which they have a defined responsibility (in our experience, other students never volunteered to read all the papers).
- *Tools*. In the RSC script (Betbeder and Tchounikine 2003), roles (which in this case are defined by the students themselves) are related to the use of specific tools at different phases of their project (Table 4).
- *Responsibilities*. In WiSim, each team member—identified by his mobile phone—is responsible for entering values for certain simulation parameters. Having access to specific functions of an application is similar—just different in granularity—to having access to different software tools as discussed in the previous point. Reciprocal

Table 3 Social matrix rotation in the UniverSanté script

$Phase_i$	Switz.	Lebanon	Tunisia	$Phase_j$	Cancer	Diabetics	Alcohol
Cancer	Amy	Andrea	Alberto	Switz.	Amy	Bernie	Marie
Diabetics	Bernie	Brigitte	Bruno	Lebanon	Andrea	Brigitte	Michelle
Alcohol	Marie	Michelle	Marco	Tunisia	Alberto	Bruno	Marco

Table 4 The roles *X* resources matrix

Roles/ documents	Paper 1	Paper 2	Paper 3	Paper 4	Roles/tools	Email forum	MS project	SPSS	Transana
Piaget	*X*	*X*			Coordinator	*X*	*X*		
Vygostky				*X*	Statistician	*X*		*X*	
Weinberger			*X*		Dialogue analyst	*X*			*X*

Tutoring (Palincsar and Brown 1984) is a script according to which students are alternatively assigned the role of 'reader' and 'questioner'; while the former reads a paragraph, the latter poses predefined questions.

Other scripts rely on "natural" differences between students, i.e. differences that are not generated by the script. Natural differences occur in two situations:

- Natural differences are known a priori when the class consists of different audiences, i.e. students with different backgrounds. For instance, the script developed by Rummel and Spada (2007) forms pairs comprising students of medicine and psychology, i.e. student roles naturally resulting from their respective background knowledge. In the UniverSanté script, medical students came from different countries (Switzerland, Lebanon, Tunisia and Cameroon), which was highly relevant given that they were required to discuss public health issues and that health problems as well as solutions (e.g. prevention) vary across countries. Many "natural" differences can be exploited on the basis of geographical differences (climate, etc.), social differences, urban versus country contexts, etc.
- When natural differences exist within a single audience, e.g. 'natural' differences of opinions among students of the same class, these can be captured by software and utilized in the same way as roles. This is the case in the ArgueGraph script in which the recording of student opinions in Phase 1 is used for the subsequent forming of groups. Gijlers and de Jong (2005) use the different hypotheses produced by students as 'natural' differences in an inquiry-based learning environment.

We do not propose a classification of resources since this has been carried out elsewhere and has led to various standards (Friesen 2005). We would like to emphasize that script mechanics rely on connections between people and resources and that these connections are often expressed using the concept of roles.

Products

Individuals, groups and classes produce objects at all phases of a script. These products range from simple data to complex multimedia objects. All efforts to generalize the description of products should rely on existing standards for metadata. This is easier to apply to large than to small objects. Larger objects (long texts, software pieces, etc.) are usually stored in an explicit way. Smaller objects comprise data such as "Olga selected Answer 2 for Question 5" in ArgueGraph or "Group 5 placed Concept 2 in cell [3 4]" in ConceptGrid. Within a clean modular architecture, these data remain 'internal' to the software tools (a chat tool, a whiteboard, a questionnaire etc.) that are included in the script as black boxes. This high modularity facilitates script authoring but inhibits scripting mechanisms which require

the reuse or processing of data that are internal to other components (for instance, using questionnaire data for the purpose of group formation as in ArgueGraph).

Resources and products can be distinguished by the fact that resources are predefined, i.e. they exist before the script is run, while products result from the unfolding script. However, this distinction is not very robust since the product of a given phase may become a resource for activities in a later phase.

The specificity of CSCL scripts is that all products are associated with a social entity, i.e. an individual, a group or a class. The ownership of products makes it possible to aggregate/dissociate products according to social plane.

Operators

A script is a workflow, i.e. a time-based sequence of operators that transform data structures, the social structure (student, groups, classes, roles) and object structures (resources and products). Each operator produces the phase_{N+i} matrices from the phase_N matrices. Different types of operators exist:

- [Products Matrix → Social Matrix] The group formation operator in ArgueGraph uses the products matrix from Phase 1 to produce the social matrix of Phase 3.
- [Social Matrix → Resources Matrix] In ArgueGraph, simple aggregation operators generate pairs in Phase 3 using the answers provided by members in Phase 1 and provide the teacher (Phase 4) with all answers to a specific question. In ConceptGrid, aggregation is used to provide the teacher with the cross-grid view required for conducting the debriefing session (Phase 5).
- [Products Matrix → Products Matrix] Aggregation is used in WiSim to turn individual parameter values provided by each member of the group into the parameter set required for running the simulation. Conversely, WiSim uses a distribution operator in Phase 6 to provide group members with fragments of the simulation results, different views of the same results or results produced by different models.
- [Social Matrix → Social Matrix] Some Jigsaw scripts and the UniverSanté script (Berger et al. 2001) apply a 90 degree rotation operator, i.e. forming groups using the rows or columns of the role *X* group matrix as illustrated in Table 3.

Script handicap

As expressed by the SWISH principles, the distribution of resources and roles among team members determines the effort to be exerted by students in order to compensate for this distribution, i.e. the intensity of the interactions required for completion of the joint activities. A script artificially increases the effort necessary to achieve a joint product, as depicted in Fig. 6, where Δ_1 represents the effort that pairs would have to naturally exert to reach a consensus, Δ_2 represents the increased effort pairs would have to exert due to the script design and Δ_3 represents the degree of divergence remaining upon task completion. In a script such as ArgueGraph, the value of Δ_1 mainly depends on the script session, i.e. the natural heterogeneity of the class, while the difference between Δ_1 and Δ_2 is an intrinsic feature of the script. This difference corresponds to script effects, not in terms of learning outcome but rather in terms of the need for interaction. Metaphorically speaking, the deeper the valley, the more effort the team has to exert in order to reach the summit on the right hand side. As mentioned earlier, if the valley is too deep, climbing may become intractable.

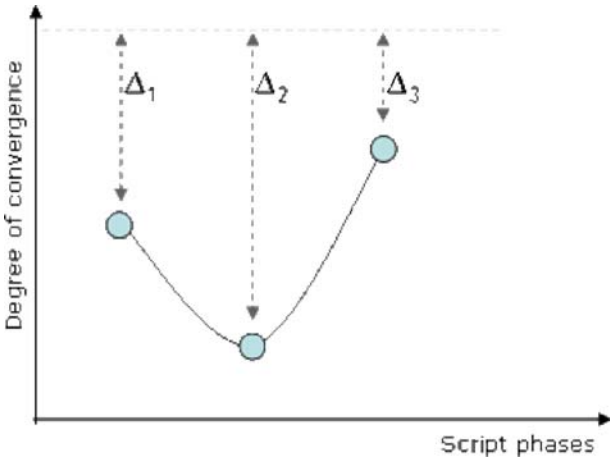

Fig. 6 Natural divergence (Δ_1) is increased by the script (Δ_2) and is to be compensated for by the team (Δ_3)

Let us use a simple metric for Δ_1, Δ_2 and Δ_3, a disagreement rate ranging from 0 to 100%, where 0% refers to unrealistic perfect agreement. The value of Δ_2 corresponds to what we refer to as 'optimal collaborative effort'. The value of Δ_3 decreases if the joint activity in phase 3 forces team members to form a joint understanding, but there is of course no activity that can force them to reach a perfectly shared understanding.

Since scripts span different social planes, Δ can be computed in different ways: the degree of disagreement between the individuals within a group Δ^{IG}, the degree of disagreement between the individuals within a class Δ^{IC} and the degree of disagreement between the groups in a class Δ^{GC}. While collaborative scripts and this paper focus on Δ^{IG}, Δ^{IC} and Δ^{GC} also have to be considered in integrated scripts given their relationship to the richness of class-wide activities, in particular to the debriefing sessions. In the following sub-sections, we use these parameters to describe how SWISH mechanisms are instantiated in the ArgueGraph and ConceptGrid examples.

SWISH mechanisms in ArgueGraph

The initial degree of disagreement (Δ_1^{IC}) in ArgueGraph can be estimated by comparing the answers to the individual questionnaire (phase 1). Let us consider a class of six students, each of whom answers a single question with one of six possible answers. Disagreement is 0% if all students provide the same answer and 100% if each student provides a different answer. With an actual class of 20 students and questions with four answers, the value of Δ_1^{IC} will of course somewhere between these extreme values. We simply compute Δ_1^{IC} as follow: for each answer to each question, we compute the percentage of other student who gave a different answer and then compute the average percentage.

We recently ran an ArgueGraph session with 14 doctoral and postdoctoral fellows at EPFL. The questionnaire included ten questions, each of which had between two and four possible answers. The value Δ_1^{IC} was 47%. The rate of disagreement ranged from 78% (for a critical question with four possible answers) to 0% (for a question that was too obvious or too politically correct and had only two answers). It is important to design questions that lead students to express divergent opinions in Phase 1; in our experience, questions with a politically correct flavour fail to produce sufficient divergence. We also examined whether individuals systematically had a different opinion from the rest of the class: This personal rate of disagreement was surprisingly stable, ranging from 48 to 58%.

Once groups have been formed by the script, we may compute the rate of disagreement within each pair (Phase 2) and average them: in the previously mentioned ArgueGraph session, the value of Δ_2^{IG} was 59%. Our pair matching algorithm is actually more subtle than such simple percentage calculations expressed as Δ_1 and Δ_2. For instance, on a scale "never," "rarely," "sometimes," "often" and "always," the answers "often" and "sometimes" should be treated as being closer to each other than "often" and "never." Therefore answers in ArgueGraph are differentially weighted, but we nonetheless use this disagreement rate as a simple appraisal of the collaborative effort that illustrates the SWISH principles and is reusable in many contexts.

At the end of their argumentation (Phase 3), pairs are required to provide a single answer to each question. The computable value of Δ_3^{IG} is therefore 0%, although this obviously does not reflect the actual level of disagreement: pairs may 'accept' a common answer without truly agreeing (Baker 1995), because they are polite or tired, or they may have the "illusion of a shared understanding" (Ross et al. 1977; Cherubini and van der Pol 2005). It is even possible, although it can not be verified, that the engagement often observed in the debriefing session (Phase 5) results from the frustration felt by some students in being forced to make choices that do not reflect their opinion in a satisfactory way. For the sake of interest, we may examine whether groups converge more in Phase 3 than individuals in Phase 1. We compute Δ_3^{GC} in the same way as Δ_1^{IC} but by considering group answers instead of individual answers: the value of Δ_3^{GC} is 51%.

Not only do answers provided by students in a questionnaire not perfectly match their actual opinions, but in addition, their divergence of opinions does not accurately predict students' engagement in argumentation. The intensity of argumentation in teams (Phase 3) does not exclusively depend on Δ_2^{IG}, but also on the personality of the students (some intensively defend their original individual answer) and on status differences (the class included, for instance, fresh PhD students as well as more experienced scholars). Similarly, the intensity of argumentation in debriefing (Phase 5) not only depends on Δ_3^{GC} but also on other factors including, for example, the timing of the whole sequence (Dillenbourg and Jermann 2007). In other words, we do not consider the rate of disagreement to be a cognitive measure, but rather a behavioural indicator of script mechanics: SWISH mechanisms can be implemented as operators that transform script data structures (social matrix, products matrix and resources) between phases in order to increase Δ^{IG}.

SWISH mechanisms in ConceptGrid

We can apply the same type of analysis to the ConceptGrid. We ran four sessions of the ConceptGrid in a Master's course on Computer-Supported Cooperative Work. Each session involved between eight and ten groups of three students working on sets of nine concepts. Grid dimensions were 4×4 with the exception of the third session, in which the grid was 3×3.

An estimate value of Δ^{IG} can be computed when concepts have been individually defined by students. The products matrix (Table 5) stores who defined which concept. In Table 5, results are aggregated across groups. A 1 is inserted in $cell_{i,j}$ if the $concept_i$ has been defined by $role_j$. The number in $cell_{i,j}$ corresponds to the number of times the $concept_i$ has been defined by $role_j$. Table 5 shows that the distribution of concepts across roles varies across teams. Some concepts (e.g. concepts 2 and 5 in grid 1) were defined by the same role in all teams. Most concepts are rather clearly associated with a role. Nonetheless, some concepts were defined by several roles (e.g. concepts 8 and 9 in grid 4). The role specificity of $concept_i$ in $grid_g$ can be computed as the standard deviation of $column_i$ in $grid_g$. If we average role specificity across all concepts in each grid, we observe that this value

Table 5 Ownership of definition according to role for each of the 4 sessions

GRID 1	Concept 1	Concept 2	Concept 3	Concept 4	Concept 5	Concept 6	Concept 7	Concept 8	Concept 9
Role 2	1	0	1	0	1	6	6	6	0
Role 1	0	0	8	9	9	1	2	1	0
Role 3	9	10	1	1	0	2	1	2	10
GRID 2	Concept 1	Concept 2	Concept 3	Concept 4	Concept 5	Concept 6	Concept 7	Concept 8	Concept 9
Role 2	1	1	2	3	1	2	7	7	3
Role 1	8	8	0	0	0	7	1	1	6
Role 3	0	0	7	6	7	0	0	0	0
GRID 2	Concept 1	Concept 2	Concept 3	Concept 4	Concept 5	Concept 6	Concept 7	Concept 8	Concept 9
Role 1	0	7	0	6	5	1	0	0	3
Role 2	6	1	5	0	2	5	0	1	1
Role 3	1	0	2	2	1	1	8	7	4
GRID 2	Concept 1	Concept 2	Concept 3	Concept 4	Concept 5	Concept 6	Concept 7	Concept 8	Concept 9
Role 2	0	7	0	6	3	1	3	4	3
Role 1	5	1	1	1	3	5	2	2	1
Role 3	2	0	6	1	0	1	1	2	4

The number of groups involved in each session was 10, 9, 8 and 8, respectively

constantly decreased throughout the fours session of ConceptGrid in the given Master's course (Fig. 7). The log files from these experiments do not allow us to discriminate the main author of a concept from the final author (somebody who might simply have edited the definition); hence this loss of specificity could also indicate a growing trend to edit the definitions produced by peers. In a paper questionnaire, 47% of students said they sometimes edited the definition produced by their partner. Nonetheless, if concept–role specificity could be computed accurately, it would serve as a good estimate of Δ_4^{IG} prior to students' construction of the grid (Phase 4).

The key phase in ConceptGrid is the phase in which groups construct the grid, or more precisely, in which two concepts defined by different roles are juxtaposed on the grid. Team members have to explain these concepts to one another in order to write a short text describing the relationship between the two concepts. Of course, if one student reads about concepts C_1, C_2 and C_3, it is natural that (s)he juxtaposes these concepts on a grid. In the first

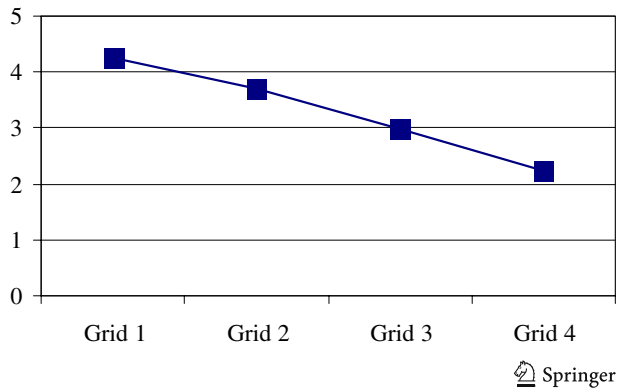

Fig. 7 Evolution of concept–role specificity across the four ConceptGrid sessions

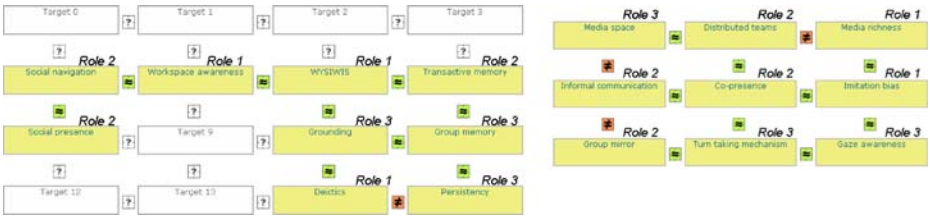

Fig. 8 Juxtaposing two concepts requires elaborate explanation when the concepts have been defined by two different students

session, for instance, the concepts "intellectual capital" and "knowledge management"—both of which had been defined by the same role in all groups—were juxtaposed on the grid by nine of the ten groups. Extra effort is required whenever students juxtapose concepts that have been defined by different authors.

In Fig. 8, the grids have been annotated in such a way as to allow identification of the role players who defined each concept. Let us refer to the relationships between concepts defined by different authors as "cross-author". In the grid on the left, six of a total of ten relationships defined by the team were cross-author relationships, and 7 of a total of 12 in the grid on the right. The value of Δ_5^{IG} can be simply estimated by the number of cross-author relationships: the higher this number, the greater the explanation effort required to construct the grid.

The value of Δ_5^{IG} does not depend on the script per se, but rather on the specific script instance. In Session 2, for example, most teams (eight of nine) juxtaposed "group memory" and "transactive memory" despite the fact that these two concepts were in each case defined by different team members. These two concepts thus have high affordance of social interactions. The estimation of Δ_5^{IG} could be refined by not only counting the number of cross-author relationships, but also by weighting each relationship with the concept–role specificity: if $role_x$ has to explain $concept_i$ to $role_y$, the explanation will have to be more elaborate if $concept_i$ is highly specific to $role_x$ than if it is also sometimes defined by $role_y$.

Of interest in the debriefing phase (Phase 6), is that many different relationships have been defined by different teams. This is captured by Δ_6^{GC}, i.e. the diversity of the grids constructed by the groups, as illustrated in Fig. 9.

As was the case for the ArgueGraph, the SWISH mechanisms that drive ConceptGrid are embedded in simple operators that relate three data structures: the social matrix (which student has which role), the individual products matrix ($concept_i$ is defined by $role_x$/$team_t$) and the group products matrix ($concept_i$ is juxtaposed with $concept_y$). The product of these

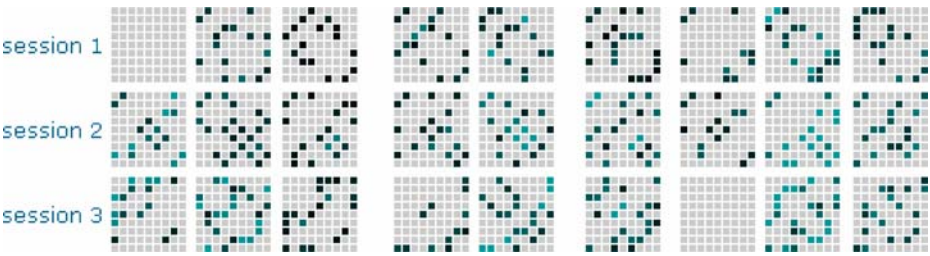

Fig. 9 Concept relationship matrices expressed by each group. A *dot* appears in $cell_{ij}$ when the $concept_i$ has been juxtaposed to $concept_j$. The *upper left matrix* is empty because this group did not clearly understand the task in constructing the first grid

matrices allows the computation of data such as the concept–role specificity and other above-mentioned variables.

Conclusions

SWISH is not a cognitive model: it does not explain why students learn through argumentation, explanation or mutual regulation. The cognitive effects of inducing such interactions have been established by other researchers. SWISH is rather a pedagogical design model, i.e. a set of principles that designers may apply to help scaffold specific classes of interaction. As is true of most pedagogical principles, these principles are not deterministic: the actual induction of target interactions depends on numerous parameters (e.g. the personality of both students and teachers), besides the script itself. Run-time adjustments are required: teachers must permanently regulate scripts. While constructivism has often been confused with "teacherlessness," our socio-constructivist scripts assign the teacher the central role of orchestrating multi-level activities.

The specificity of these principles lies in the fact that their operationalization relies on mechanisms for the transformation of data structures (social matrix, resources matrix and products matrix) that are handled by the different activities of the script. This set of mechanisms, which we refer to as the "mechanics" of scripts, and the SWISH model constitute the core difference between a genuine lesson plan and a CSCL script. Unfortunately, these mechanisms also represent an obstacle in the abstraction process that is required for developing authoring tools.

Acknowledgments This work was partly supported by the Swiss Center for Innovation in Learning (SCIL, University of St. Gallen) which is funded by the Gerbert Rüf foundation. Special thanks to SCIL members Taiga Brahm and Sabine Seufert. The paper was also supported by Kaleidoscope, a European Network of Excellence, in particular members of the MOSIL and COSSCILE groups, including Frank Fischer, Lars Kobbe, Armin Weinberger, Andreas Harrer, Nils Malzahn, Paivi Hakkinen, Raia Hamalainen, Sanna Jarvela, Ulrich Hoppe and Pierre Tchounikine. We also benefited from the contributions of our team members Shuja Parvez and Patrick Jermann and our former colleague Daniel Schneider.

References

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

Ayala, G., & Yano, Y. (1998). A collaborative learning environment based on intelligent agents. *Expert Systems with Applications*, *14*, 129–137.

Baker, M. J. (1995). Negotiation in collaborative problem-solving dialogues. In R.-J. Beun, M. J. Baker, & M. Reiner (Eds.), *Dialogue and instruction* (pp. 39–55). Berlin: Springer.

Baker, M. J., & Lund, K. (1996). Flexibly structuring the interaction in a CSCL environment. In P. Brna, A. Paiva, & J. Self (Eds.), *Proceedings of the European conference on artificial intelligence in education* (pp. 401–407). Lisbon, Portugal: Edicoes Colibri Sept. 20–Oct. 2.

Barros, B., & Verdejo, M. F. (2000). Analysing student interaction processes in order to improve collaboration. The DEGREE approach. *International Journal of Artificial Intelligence in Education*, *11*, 221–241.

Berger, A., Moretti, R., Chastonay, P., Dillenbourg, P., Bchir, A., Baddoura, R., et al. (2001). Teaching community health by exploiting international socio-cultural and economical differences. In P. Dillenbourg, A. Eurelings, & K. Hakkarainen (Eds.), *Proceedings of the first European conference on computer supported collaborative learning* (pp. 97–105). Maastricht, March 2001.

Betbeder, M. L, & Tchounikine, P. (2003). Symba: A framework to support collective activities in an educational context. *In Proceedings of the international conference on computers in education* (pp. 188–196), Hong Kong.

Blaye, A., Light, P., Joiner, R., & Sheldon, S. (1991). Collaboration as a facilitator of planning and problem solving on a computer based task. *British Journal of Psychology*, *9*, 471–483.

Brousseau, G. (1998). *Théorie des situations didactiques*. Grenoble: La Pensée Sauvage.
Cherubini, M., & van der Pol, J. (2005). Grounding is not shared understanding: Distinguishing grounding at an utterance and knowledge level. *In CONTEXT'05, the fifth international and interdisciplinary conference on modeling and using context*, 2005.
Clark, H. H., & Wilkes-Gibbs, D. (1986). Referring as a collaborative process. *Cognition*, *22*, 1–39.
Constantino-Gonzalez, M., & Suthers, D. (2000). A coached collaborative learning environment for Entity-Relationship modeling. *In Proceedings of the 5th international conference on intelligent tutoring systems* (pp. 324–333). Montreal: Canada.
De Jong, T., & van Jooligen (1998). Scientific discovery learning with computer simulations of conceptual domains. *Review of Educational Research*, *68*, 179–202.
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., Baker, M., Blaye, A., & O'Malley, C. (1996). The evolution of research on collaborative learning. In E. Spada & P. Reiman (Eds.), *Learning in humans and machine: Towards an interdisciplinary learning science* (pp. 189–211). Oxford: Elsevier.
Dillenbourg, P., & Fischer, F. (2007). Basics of computer-supported collaborative learning. Zeitschrift für Berufs-und Wirtschaftspädagogik, *21*, 111–130.
Dillenbourg, P., & Jermann, P. (2007). Designing integrative scripts. In F. Fischer, H. Mandl, J. Haake, & I. Kollar (Eds.), *Scripting computer-supported collaborative learning—Cognitive, computational, and educational perspectives* (pp. 275–301). New York: Springer Computer-supported Collaborative Learning Series.
Dillenbourg, P., Ott, D., Wehrle, T., Bourquin, Y., Jermann, P., Corti, D., et al. (2002). The socio-cognitive functions of community mirrors. In F. Flückiger, C. Jutz, P. Schulz, & L. Cantoni (Eds.), *Proceedings of the 4th international conference on new educational environments*. Lugano, May 8–11, 2002.
Dillenbourg, P., & Tchounikine, P. (2007). Flexibility in macro CSCl scripts. *Journal of Computer Assisted Learning*, *23*(1), 1–13.
Dillenbourg, P., & Traum, D. (2006). Sharing solutions: persistence and grounding in multi-modal collaborative problem solving. *Journal of the Learning Sciences*, *15*(1), 121–151.
Doise, W., & Mugny, G. (1984). *The social development of the intellect*. Oxford: Pergamon.
Friesen, M. (2005). Interoperability and learning objects: An overview of E-learning standardization. *Interdisciplinary Journal of Knowledge and Learning Objects*, *1*, 23–30.
Gijlers, H., & de Jong, T. (2005). Confronting ideas in collaborative scientific discovery learning. Paper presented at AERA 2005, Montreal, CA.
Hoppe, U. H., & Ploetzner, R. (1999). Can analytic models support learning in groups? In P. Dillenbourg (Ed.), *Collaborative-learning: Cognitive and computational approaches* (pp.147–168). Oxford: Elsevier.
Inaba, A., & Okamoto, T. (1996). Development of the intelligent discussion support system for collaborative learning. *In Proceedings of ED-TELECOM'96* (pp. 137–142). Boston, MA.
Inaba, A., Supnithi, T., Ikeda, M., Mizoguchi, R., & Toyoda, J. (2000). How can we form effective collaborative learning groups? *In Proceedings of the 5th international conference on intelligent tutoring systems* (pp. 282–291), June 19–23, 2000.
Jermann, P., & Dillenbourg, P. (2003). Elaborating new arguments through a CSCL scenario. In G. Andriessen, M. Baker, & D. Suthers (Eds.), *Arguing to learn: Confronting cognitions in computer-supported collaborative learning environments* (pp. 205–226). Amsterdam: Kluwer CSCL Book Series.
Jermann, P., & Dillenbourg, P. (2007). Group mirrors to support interaction regulation in collaborative problem solving. *Computers and Education,* in press.
Jermann, P., Dillenbourg, P., & Brouze, J. C. (1999). Dialectics for collective activities: An approach to virtual campus design. *In Proceedings of the 9th international conference on AI in education*. Le Mans, France, July 1999.
Jermann, P., Soller, A., & Mühlenbrock, M. (2001). From mirroring to guiding: A review of the state of art technology for supporting collaborative learning. *In Proceedings of Euro-CSCL* (pp. 324–331). Maastricht, NL.
Kobbe, L., Weinberger, A., Dillenbourg, P., Harrer, A., Hämäläinen, R., & Fischer, F. (2007). Specifying computer-supported collaboration scripts. *International Journal of Computer-Supported Collaborative Learning*, *2*(2–3), 211–224.
McManus, M., & Aiken, R. (1995). Monitoring computer-based problem solving. *Journal of Artificial Intelligence in Education*, *6*(4), 307–336.
Miyake, N. (1986). Constructive interaction and the iterative process of understanding. *Cognitive Science*, *10*, 151–177.
Muehlenbrock, M. (2006). Learning group formation based on learner profile and context. *International Journal on E-Learning*, *5*(1), 19–24.

O'Malley, C. (1987). *Understanding explanation. Cognitive science research report no. CSRP-88*. Great Britain: University of Sussex.

Palincsar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and Instruction*, *1*(2), 117–175.

Randolph, C. H., & Evertson, C. M. (1994). Images of management in a learner-centered classroom. *Action in Teacher Education*, *16*(1), 55–65.

Ronen, M., Kohen-Vacs, D., Raz-Fogel, N. (2006). *Structuring, sharing and reusing asynchronous collaborative pedagogy.* In *International conference of the learning sciences*. Bloomington, IN: Indiana University.

Roschelle, J. (1990). Designing for conversations. Paper presented at the AAAI Symposium on knowledge-based environments for learning and teaching. Standford, CA, March 1990.

Roschelle, J., & Teasley, S. D. (1995). The construction of shared knowledge in collaborative problem solving. In C. E. O'Malley (Ed.), *Computer-supported collaborative learning* (pp. 69–197). Berlin: Springer.

Ross, L., Greene, D., & House, P. (1977). The false consensus phenomenon: An attributional bias in self-perception and social perception processes. *Journal of Experimental Social Psychology*, *13*, 279–301.

Rummel, N., & Spada, H. (2007). Can people learn computer-mediated collaboration by following a script. In F. Fischer, H. Mandl, J. Haake, & I. Kollar (Eds.), *Scripting computer-supported communication of knowledge. Cognitive, computational, and educational perspectives*. New York: Springer CSCL Book Series.

Schwartz, D. L. (1995). The emergence of abstract dyad representations in dyad problem solving. *The Journal of the Learning Sciences*, *4*(3), 321–354.

Soller, A. L. (2001). Supporting Social Interaction in an Intelligent Collaborative Learning System. *International Journal of Artificial Intelligence in Education*, *12*(1), 40–62.

Suthers, D. (1999). Representational bias as guidance for learning interactions: A research agenda. In *Proceedings of the 9th world conference on artificial intelligence in education (AIED'97)* (pp. 121–128). Le Mans, France, July 19–23, 1999.

Veerman, A. L., & Treasure-Jones, T. (1999). Software for problem solving through collaborative argumentation. In P. Coirier & J. E. B. Andriessen (Eds.), *Foundations of argumentative text processing* (pp. 203–230). Amsterdam: Amsterdam University Press.

Wasson, B. (1998). Identifying coordination agents for collaborative telelearning. *International Journal of Artificial Intelligence in Education*, *9*, 275–299.

Weinberger, A., Fischer, F., & Mandl, H. (2002). Fostering computer supported collaborative learning with cooperation scripts and scaffolds. In G. Stahl (Ed.), *Computer support for collaborative learning: Foundations for a CSCL community. Proceedings of the conference on computer support for collaborative learning* (pp. 573–574). Boulder, CO.

Wessner, M., & Pfister, H. (2001). Group formation in computer-supported collaborative learning. *In Proceedings of the 2001 international ACM SIGGROUP conference on supporting group work*. Boulder, Colorado, USA, September 30–October 03, 2001.