T. Koschmann, D. Suthers, & T.W. Chan (Eds.), *Computer Supported Collaborative Learning 2005: The Next 10 Years!*
Mahwah, NJ: Lawrence Erlbaum Associates. ©2005 International Society of the Learning Sciences.

CSCL Scripts: Modelling Features and Potential Use

Yongwu Miao, Kay Hoeksema, Heinz Ulrich Hoppe, Andreas Harrer
Institute of Computer Science and Interactive Systems
University of Duisburg-Essen
(miao, hoeksema, hoppe, harrer)@collide.info

Abstract. The design of collaboration scripts is a new focus of research within the CSCL community. In order to support the design, communication, analysis, simulation and even execution of collaboration scripts, a general specification language to describe collaboration scripts is needed. In this paper, we analyse the suitability and limitations of IMS LD for modelling collaborative learning processes. Based on the analysis, we propose a CSCL scripting language. This paper presents the conceptual framework of this modelling language and the underlying design ideas. Furthermore, two developed CSCL script authoring tools are briefly described. Finally, we discuss potential types of usage and system support possibilities of CSCL scripts.

Keywords: IMS LD, CSCL scripting language, CSCL script, CSCL script authoring tool

INTRODUCTION

According to O'Donnell & Dansereau (1992) a collaboration script is a set of instructions specifying how the group members should interact and collaborate to solve a problem. The term "script" was initially used in schema theory by Schank and Abelson (1977). According to schema theory, a script is a mental structure representing the people's knowledge about actors, objects, and appropriate actions within specific situations. When group members interact with each other, a shared script can help them to reduce the uncertainty about coordination efforts (Mäkitalo et al., 2004), because they know how to behave and what to expect in particular situations. By providing learners with a collaboration script, it is conceivable that learners can also aim at cognitive objectives like fostering understanding or recall. Additionally, collaboration scripts might also support the development of meta-cognitive, motivational, or emotional competencies (Kollar et al. 2003). A collaboration script is normally represented in the learners' minds (internal representation) and can be represented somewhere in the learning environment (external representation). Because we focus on using collaboration scripts in computer settings, we are interested in representing collaboration scripts in a formal way so that they can be handled by the computer. Such a computational representation of a collaboration script is called a CSCL script.

The conceptual components of a collaboration script and their relations have been discussed in literature (Dillenbourg 2002, Kollar et al., 2003). However, a general modelling language for formalising collaboration scripts is still missing. Furthermore, there is no corresponding authoring tool for CSCL practitioners to create, reuse, integrate, and customise CSCL scripts without a high overhead of technical knowledge. As a first step in the direction of a CSCL scripting language we investigate existing learning process modelling languages. The most important attempt in the current discussion in this direction is IMS Learning Design (IMS LD), a standard published by the IMS consortium based on the Educational Modelling language (EML) developed by the Dutch Open University OUNL (Koper, 2001). It is claimed that IMS LD can formally describe any design of teaching-learning processes for a wide range of pedagogical approaches (Koper, 2001, Koper & Olivier 2004). This modelling language has strengths in specifying personalised learning and asynchronous cooperative learning. However, IMS LD provides insufficient support to model group-based, synchronous collaborative learning activities. Caeiro et al. (2003) criticised IMS LD regarding CSCL purposes and suggested a modification and extension of the specification. This modification and extension is just restricted in role-part and method part. Hernandez et al. (2004) suggested adding a special type of service, called "groupservice" to extend the capacity of IMS LD. Such an extension at service level, rather than at activity level, cannot appropriately capture the characteristics of collaborative learning activities.

The research work presented in this paper aims at developing a scripting language for formalising CSCL scripts and exploring their potential types of usage and system support possibilities. In the first part of this paper, we intend to clarify the limits of IMS LD when working on a computational methodology for collaborative learning process scripting. Based on the analysis, we propose a scripting language to facilitate modelling collaborative learning processes. Rather than a systematic description of the CSCL scripting language, we present it by focusing on how the identified problems of IMS LD for CSCL scripts are solved. Then we briefly describe two script authoring tools based on the CSCL scripting language. In the second part of this paper we try to address the current misbalance between the broad acceptance of CSCL scripts as being an innovative and relevant topic of research into learning technologies, and the weak definition of potential forms of usage of such representations in learning situations. The most frequent answer to this question is a CSCL script as a source for a configuration tool to support runtime environments during the learning process. As we explain in our second part, such a scripting language will be helpful for CSCL practitioners (e.g., teachers and students) in design phase (e.g. editing, communicating, predicting, simulating) and in the execution phase (e.g. configuration, monitoring, scaffolding).

INVESTIGATING THE CAPACITY OF IMS LD FOR FORMALISING COLLABORATIVE LEARNING SCRIPTS

A collaborative learning experience can be described by a collaboration script. Many collaboration scripts have been designed, tested, and even embedded in CSCL applications (e.g., Hoppe & Ploetzner 1999, Guzdial & Turns 2000, Miao et al. 2000, Pfister & Mühlpfordt 2002). The European MOSIL project (MOSIL, 2004) worked on generalised CSCL scripts. One example discussed in the MOSIL project is the *maze* script, which is described in detail in (Jansen et al., 2004). In this script student groups try to develop strategies how to escape out of an arbitrary maze. A typical use case of the script is described below.

First the teacher gives a short introduction about the structure of the learning task and then divides the class (24 students) into 6 groups of four (activity 1). Toni and Darina form group 1 together with two other students. They can use a little lego robot in a physical maze and a computer simulation with a robot in a maze. Toni and Darina are the “strategy developers” and the other two students are the “maze builders” to test the strategies (so called rule sets). Strategies and mazes can be stored on a central server (activity 2). Then each group in turn presents their developed group results. As a group speaker Toni presents their first results (activity 3). Now all groups are allowed to access the other groups work results. The roles within the group have switched, so Toni and Darina now are the “maze builders” trying to improve the groups’ mazes so the other groups’ strategies will not help to escape out of them. Meanwhile the other two students are working as the “strategy developers”. A competition has started to find out which group develops the best strategies and mazes (activity 4). Finally the group that wins in the competition shows their achieved results (activity 5). This script can be applied when there is a thesis – antithesis (rule set- against-maze or pro-against-contra-argumentation) situation which can build the basis for a competition between groups. In the following discussion we will refer to the features of the described script.

When using IMS LD to formalise collaboration scripts, we see several major difficulties and challenges:

- Modelling groups
- Modelling artefacts
- Modelling dynamic features
- Modelling complicated control flow
- Modelling varied forms of social interaction

This will be discussed in more detail below by referring to the maze script explained above.

1) Modelling group work with IMS LD confronts with the problem how to model multiple groups with the same role and the dynamic changes of groups. IMS LD enables to define multiple roles. Each role can be played by multiple persons. When investigating, we found that in many cases the notation of “role” can be used to model groups for CSCL scripts. However, by using IMS LD it is very difficult to specify how a group work pattern is assigned to several groups working in parallel and how sub groups can be defined within these groups (like needed in activity 2 and 4). If each group/subgroup is defined as a role, the designer has to define a list of roles representing multiple groups (e.g., maze builder 1, …, maze builder 6). The problem of this solution is that the number of groups in a run is inpredictable during the modelling phase. If only one role (e.g., maze builder) is defined for all subgroups building mazes, then all persons who build mazes will have the same role. The problem of this solution is the information about groups/subgroups will miss and the run-time system cannot support inter-/intra-group collaboration appropriately. In addition, in IMS LD roles are assigned to persons before running a unit of learning and these assignments keep unchanged within the life cycle of the run. However, in some situations (e.g., in activity 1) groups are formed and group members are assigned after the start of the process execution. Therefore, in some situations, the notation of role cannot meet the requirement to model groups.

2) A second major difficulty while modelling CSCL scripts with IMS LD we see in modelling artefacts. In learning processes, actors usually generate artefacts such as a vote, an answer, an argument, or a design. In IMS LD, an artefact can be modelled as a property, e.g. of a person or a role, which creates the artefact. This property can be used to maintain information such as the outcome of a person or a role and to support personalised learning. In collaborative learning processes, an artefact is usually created and shared by a group of people. It is normally used as a mediation to facilitate indirect interaction among group members. It may be created in an

activity and used in other activities like an information-flow. For example, in the maze script the different sub groups produce, reuse and improve rule sets and maze definitions. In order to support group interaction, an artefact should have attributes such as artefact type, status, created_by, creation_activities, contributors, consume_activities, current_users, and so on. By using IMS LD to model an artefact as a property, one has to model all attributes of the artefact as properties as well. These properties should be defined as a property-group with many restrictions. Such a complex definition cannot be intuitively understood. It will be very difficult to model dynamic features even for technical experienced designers, because the limited data-types of properties and many references make it very complicated to handle artefacts. In addition, it is difficult to model a collective artefact, because IMS LD does not support array-like data-types for a property. For example, it is inpredictable how many rule sets are developed by a group in activity 2. Furthermore, an artefact as a maze definition may have a complex data structure and has to be handled by using specific application tools. IMS LD has no means to specify the relation between artefacts and tools.

3) A third major difficulty while modelling CSCL scripts with IMS LD occurs when modelling dynamic process aspects. IMS LD provides two categories of operations on process elements: read-access operations ("getters") to get the state of process elements (e.g., users-in-role, datetime-activity-started) and write-access operations to change the state of process elements (e.g., change-property-value, hide/showe elements, and send notification) to model dynamic features of learning processes. For modelling collaborative learning processes, more write operations are needed. For example, during the first activity, the teacher creates groups/subgroups and assigns group members. An example of needed read-operations can be found in activity 3 "number-of-role-members" which can be used as the upper limit of a loop control variable to model multiple times of presentation performed by each working group in turn. At least, process element operations concerning our proposed extensions like group and artefact should be extended.

4) A fourth major problem is how to model complex process structures. IMS LD provides *play, act, role-part,* and *activity-structure* to model structural relations at different levels. Primarily linear structured learning/teaching processes with concurrently executable activities can be modelled. However, as Caeiro et. al. (2003) pointed, the linear structure of a play with a series of *acts* introduced a great rigidity while modelling network structures. Although it is possible to model non-linear structural relations among activities by using conditions and notifications, the specification of a collaborative learning process might be very complicated and confusing. In addition, it is very difficult to model a control flow associated with complicated combination of process instance thread splitting and synchronisation. Such a situation must happen in collaborative learning processes although the control flow of the maze script can be modelled by using IMS LD.

5) The last difficulty we want to stress in this paper occurs when modelling varied forms of social interaction. IMS LD uses a metaphor of a theatrical play to model learning/teaching processes. A play consists of a sequence of acts and within an act there is a set of role-parts. These role-parts can run together in parallel. Role-parts enable multiple users, playing the same or different roles, to do the same thing or different things concurrently on the same act. For example, while each student reads the same article, the teacher prepares presentation slides. If a group of people performs a synchronous activity, IMS LS enables them to use a conference service and provides no means at activity level to support collaboration. In collaborative learning processes, it is quite usual that people with the same or/and different roles perform a shared activity through direct or indirect interaction. While making the joint effort, people with different roles may have different rights to interact with other roles and the environment. In particular, it can not be clearly modelled by using IMS LD whether and how people collaborate, because people may work in a variety of social forms: Individually, in an informal group, in sub-groups (e.g., in activity 2), in a group as a whole (e.g., in activity 1), or in a community.

AN APPROACH TO FORMALISE CSCL SCRIPTS

In order to enhance effective collaboration designs, we have developped a CSCL scripting language to formalise collaboration scripts. Because of the limited space of the paper, rather than a systematic description, we briefly present the CSCL scripting language by explaining the core concepts and their relations. Then we focus on describing how the identified problems of IMS LD for CSCL scripts are solved in our scripting language.

CSCL Scripting Language

In this subsection, we briefly present the core concepts and their relations of the CSCL scripting language.

A *CSCL script* is a specific learning design which emphases collaboration. A CSCL script contains contextual information that applies to other elements within the process. As shown in Figure 1, a CSCL script consists of a set of *roles*, *activities*, *transitions*, *artefacts*, and *environments*. A CSCL script has attributes such as learning objectives, prerequisites, design rationale, coercion degree, granularity, duration, target audience, learning context, script-specific properties, and generic information (e.g., id, name, description, status, creation date, and so on). The attribute "design rationale" enables to express and communicate the design ideas and underlying pedagogic principles. The values of the attribute “coercion degree” represent different degrees of “informedness”. CSCL scripts with different coercion degrees have different usages, which will be discussed later in the paper. If a lower-grained CSCL script is embedded in a higher-grained CSCL script, the mappings between the roles, properties, and artefacts of two CSCL scripts should be specified. A *role* is used to distinguish users who have different privileges and obligations in the processes described in the CSCL script. Both *persons* and *groups* can take a role. A group can have subgroups and person members. An *activity* is a definition about a logical unit of task performed individually or collaboratively. There are three types of activities: atomic activity, compound activity, and route activity. A compound activity is decomposable consisting of a set of networked activities and even other scripts. A *transition* specifies a temporal preceding relation between two activities. An *artefact* may be created and shared in and/or across activities as an intermediate product and/or a final outcome. An *environment* can contain sub-environments and may contain tools and contents. A *tool* may use artefacts as input parameters and/or output parameters. A *content* is a kind of learning objects which exist and are accessable. An *action* is an operation and may be performed by users during an activity or by the system before/after an activity. A *property* may be atomic or may have internal structure. An *expression* may use properties and other expressions as operands. Like IMS LD, a *condition* refers to a condition clause which is defined as if-then-else rule consisting of a logical expression and actions, transitions, and/or other conditions. Actions, properties, expressions, and conditions have very complicated relations with other process elements (e.g., scripts, roles, activities, artefacts, persons, groups, environments, and so on). For example, an action may use process elements as parameters and change the values of attributes of certain process elements. Such relations are not drawn in this diagram in order to keep the diagram simple and readable.

Using the scripting language to formalise a collaboration script means specifying how persons and/or groups, playing certain roles, work collaboratively towards certain outcomes (partially as artefacts) by performing temporally structured activities within environments, where needed tools and content are available. Actions, properties, expressions, and conditions are useful to modelling more complicated, dynamic control-flow and information-flow in collaborative learning processes.

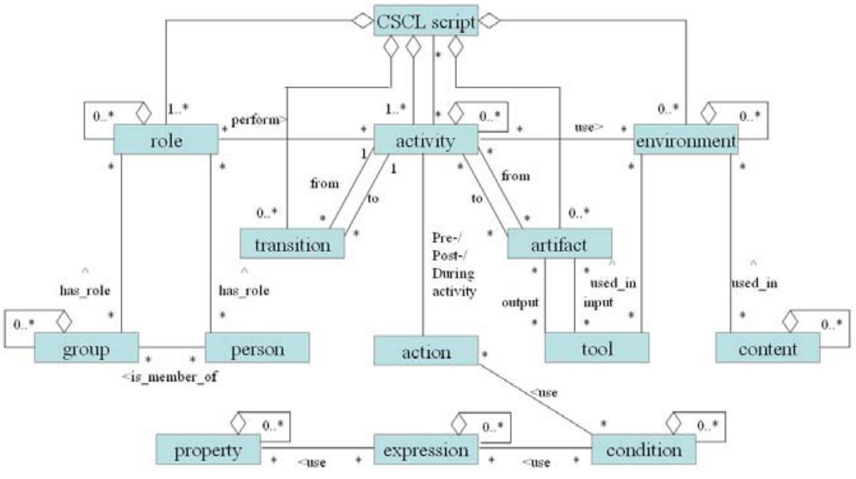


Figure 1: Core modelling elements and their interrelation

Solutions

In this subsection, we focus on presenting our solutions to the identified problems of IMS LD for CSCL scripts. We refer here to the Maze script to show the feasibility of our modelling elements to address these problems.

Explicitly introducing groups

The introduction of a group element enables to model group based collaboration intuitive and simple. In our CSCL scripting language, a group is modelled by using attributes such as name, max-size, min-size, person members, super-groups, sub-groups, engaged roles, form-policy, disband-policy, dynamic/static, and run-time information. In addition, local-/global group properties are added for capturing group characteristics. A group can be assigned to a role. Therefore, when a group role is defined like “maze builder” in maze script, the activity 2 is assigned to this role just once no matter how many groups will play this role at a run. On the one hand, a group can have subgroups and form a hierarchically structured organisation (a directed-acycle-graph). Any change in the organisation has no affect on the definition of the role in scripts. On the other hand, re-definition of roles in scripts does not effect on organisation. A question raises that when to model a group and when to use a role. From our perspective, some roles are organisation-oriented definitions like students and staff. Others are behaviour-oriented role such as meeting chairman and tutor. It would better to model an organisation-oriented role as a group role and to model a behaviour-oriented role as a person role.

Explicitly introducing artefacts

The *artefact* element does not exist in the IMS LD specification. As we explained already, the usage of artefact elements enables to model within CSCL contexts much more intuitive and easier than to model the same process within IMS LD, because some burden for designers to handle technical tasks are released by providing built-in mechanisms. In our language, an artefact, such as a Maze rule set, is treated as a file which can be a MIME-type or user-defined type. The attributes of an artefact contain generic information (e.g., title, description, type, status, URL, sharable, and aggregated), association information (e.g., creation_activities, consume_activities, and default_tool), and run-time information (e.g., created_by, creation_time, contributors, last_modification_time, current_users, locked_status, and so on). An artefact and its status will be visible in the environment of the creation-/consume- activities at run-time. The specification about the relations between artefacts and tools will help the run-time system to pass to/from artefacts as input/output parameters to tools automatically at run time. Some expressions and actions related to artefacts should be added for mediating group work such as get-current-users-of-artefact and change-artefact-status. The artefact-specific properties may be useful to model a specific feature of an artefact. As an aggregated artefact, it is possible to append collective information to the same file.

Extending actions and expressions

An *action* is a generic and powerful mechanism to model dynamic features of a collaborative learning process. Some actions are components of the CSCL scripting language that can be executed directly by the run-time system. In addition, we add an action declaration mechanism for experts to define a procedure by using the CSCL scripting language. In order to support the definition of complicated procedures, we add a "collection" data type and a loop control structure. The defined procedure can be interpreted by the run-time system into process element operations, and in turn, into executable code. Therefore, complicated actions can be defined by using an action declaration and assigning the parameters needed. IMS LD provides a limited set of actions such as property operations, showing/hiding entity, and notification. The action notation we introduced provides a unified form of operations including not only actions defined in IMS LD but also commonly used operations concerning script, activity, artefact, role, group, person, transition, environment, and their relations. An example action is enabling students to access all the other groups work results. Another example action is exchanging the role assignment between "maze builders" and "rule set developers" before the start of activity 4 in the example script. An *expression* is defined as it is in IMS LD: there some read operations can be used as operands in expressions like "is-member-of-role", "datetime-activity-started", and "complete". However, it is necessary to add read operations to support collaboration such as "is-all-role-members-online" and "artefact-contributors". Furthermore, corresponding to the action declaration, we add an expression declaration mechanism for experts to define complicated expressions which could be reused by normal teachers and students.

Introducing transitions and routing activities

We partially accept the suggestion of Caeiro et al. (2003) to introduce transitions and routing constructs recommended by the Workflow Management Coalition (WfMC home page). Because interactions of person-to-person, group-to-group, and role-to-role and splitting and synchronisation of process threads never restricted at higher levels, we have to use such a mechanism not only at play level but all possible levels in order to model the arbitrary complicated structural relations among activities.

Using activity-centred methods to assign roles

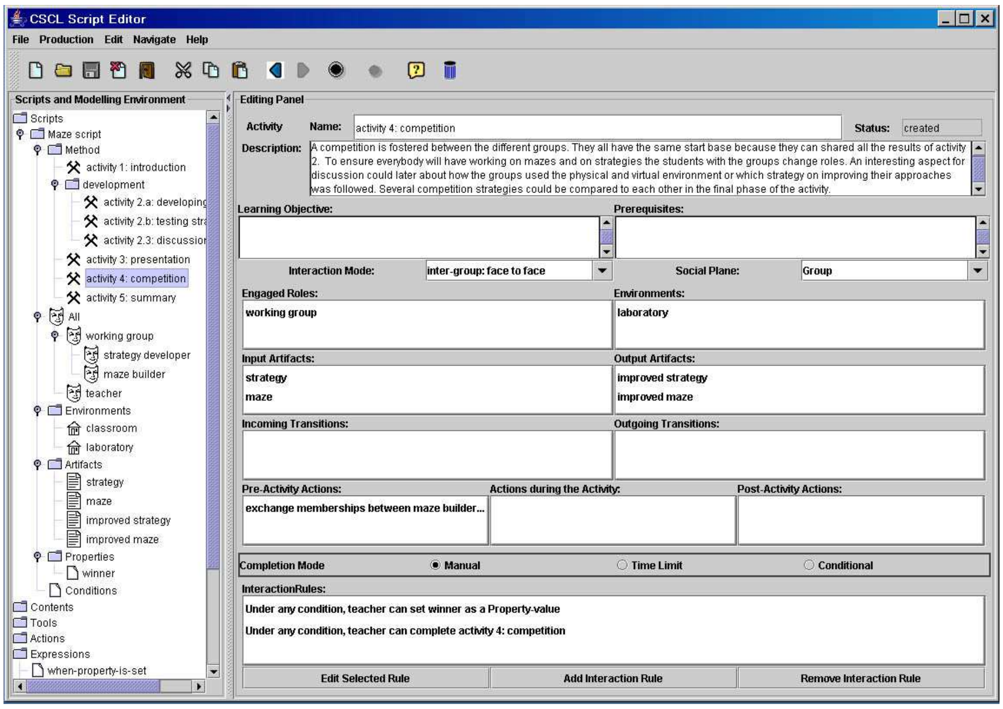
We give up the metaphor of a theatrical play and the role-part method. Instead, we use an activity centered role assignment method. In CSCL scripting language, for modelling an activity, the attributes are defined to specify engaged roles, used environments, input/output artefacts, transitions and restrictions, pre-/post-/during activity actions, user-defined activity-specific properties, completion-mode, execution-time, completion-condition, mode of interaction, social plane, interaction rules, generic information, and simulation information. Some attributes are important for designers to model collaborative processes and some for the run-time system to configure collaborative learning environments appropriately for users. For example, the possible values of social planes are: separately with a certain role, individually with a certain role, collaboratively with one and/or multiple roles, collaboratively in subgroups with a certain role, and so on. If the choice is "separately", the run-time system will create an activity instance for each user who reaches the activity. If anyone completes his activity (e.g., submitted a final rule set), all activity instances terminate. The "individually" means that the run-time system will create an activity instance for each user. The run-time system synchronises access to the following activity by continuously checking whether all users have already completed the current activity. In comparison, the run-time system based on IMS LD typically handles this situation defined by using the role-part method. The choice of "collaboratively with one and/or multiple roles" makes the run-time system create only one activity instance and a session facilitating collaboration (e.g., in activity 1). The semantics of the value "collaboratively in subgroups with a certain role" is that the run-time system creates an activity instance and a session for each sub-group and the members of each sub-group can have a shared activity workspace. The run-time system synchronises access to the next activity when all subgroups finish their work (e.g., in activity 2). Another example is the attribute "interaction rules". An interaction rule specifies under which condition which role can (not) perform which actions. For example, in activity 1, the tutor can perform the actions to create (sub)groups and assign group members. In activity 4, students can access strategies and mazes created by other groups. Such information can be used by the run-time system to automatically provide corresponding awareness information and to avoid work overload. It is of importance to provide the necessary information to the right people at right time. In short, interaction rules explicitly specify different responsibilities of different roles in a collaborative learning activity.

CSCL SCRIPT AUTHORING TOOLS

In order to support the effective formalisation of collaboration scripts, we are two tools using different representation perspectives on CSCL scripts: a tree based and a diagram based perspective. The tree-based view provides both an overview of the hierarchical structure and a view of semantic details. The diagram-based view is suitable for specifying processes with layered, more complicated control flow structure intuitively. Both tools are based on the CSCL scripting language described in the last section. They enable designers which are not technical experts to understand and design collaboration scripts. Both tools are suited to capture the essential concepts of the Maze script. This is partially shown in figures 2 & 3. The scripts can be translated from/into XML-formatted CSCL scripts automatically by the tools. A CSCL script can be created, saved, validated, and delivered. For supporting different usage purposes, the tools can adapt the user interface to enable modelling at different coercion degrees.

Tree-based Authoring Tool

The user interface of the tree-based authoring tool is shown in Figure 2. The window of the tool consists of a tool bar and two panels. The left panel is used to define the CSCL script structure and the right panel is used to create detailed designs. In the script structure panel, each CSCL script is shown as a tree in which compound activities are represented as intermediate nodes and atomic/routing activities are represented as leaves. Like IMS-LD, there is no transition between subactivities of a selection activity. However, all transitions between subactivities nested in a networked activity have to be explicitly represented. The sub-flow of a sequence activity can be represented as a sequence of activities and the order implies the transitions. Such a design is based on a fact that the main structural relations of collaborative learning scripts are sequential.



Activity Name: activity 4: competition Status: created

Description: A competition is fostered between the different groups. They all have the same start base because they can shared all the results of activity 2. To ensure everybody will have working on mazes and on strategies the students with the groups change roles. An interesting aspect for discussion could later about how the groups used the physical and virtual environment or which strategy on improving their approaches was followed. Several competition strategies could be compared to each other in the final phase of the activity.

Learning Objective:

Prerequisites:

Interaction Mode: inter-group: face to face

Social Plane: Group

Engaged Roles: working group

Environments: laboratory

Input Artifacts: strategy, maze

Output Artifacts: improved strategy, improved maze

Incoming Transitions:

Outgoing Transitions:

Pre-Activity Actions: exchange memberships between maze builder...

Actions during the Activity:

Post-Activity Actions:

Completion Mode [✓ Manual] Time Limit Conditional

InteractionRules:
Under any condition, teacher can set winner as a Property-value
Under any condition, teacher can complete activity 4: competition

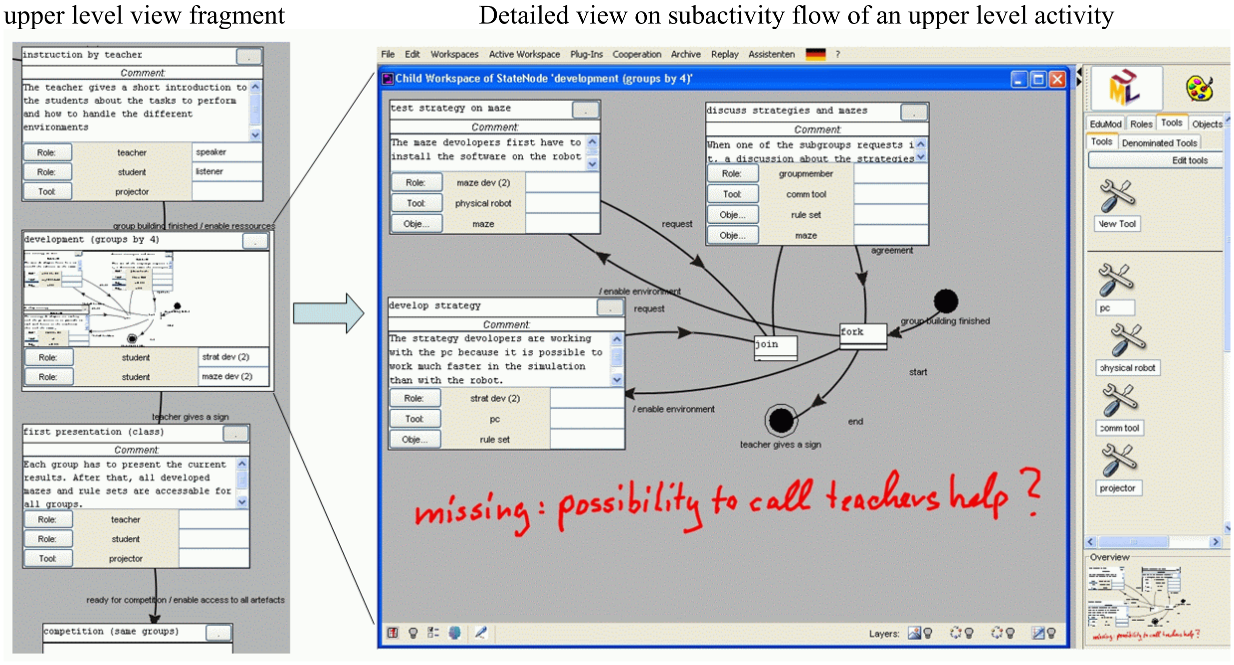
Figure 2: The user interface of the tree-based authoring tool

Figure 2 shows an example definition of the maze script. The root process structure of a script is called a method. The method of the maze script contains five activities represented in a sequence. Among these the second one is a compound activity of a type “concurrent” which contains three concurrently executable subactivities. The following sub-trees are other CSCL script components including roles, environments, artefacts, properties, and conditions. Other trees below are reusable components such as contents, tools, actions, and expressions. A user can create or load a CSCL script and then create elements such as activities, transitions, roles, environments, artefacts, and so on that make up a script. If selecting an element, the user can see and edit the detailed specification of the element. Figure 2 illustrates the definition of activity 4. The tool allows for defining the relations between elements by using drag & drop operations. For example, if assigning a role named “working group” to the activity named “activity 4”, the user can drag the tree-node representing the role from the script tree and drop it in the list of “Engaged Roles” of the activity 4.

Diagram-based Authoring Tool

The diagram-based tool is based on state chart diagrams. It supports the definition of complex control flows on arbitrary levels. Each state node corresponds to an activity and each arrow represents a transition. The nested sub-flow of an activity can be specified by drawing a new state chart in a new workspace that pops up when double clicking the activity node. The designer can create and remove elements (like a role, a tool or an artefact) by using drag & drop operations. Detailed specification of each element can be defined within a popped-up dialog window for each kind of elements.

upper level view fragment Detailed view on subactivity flow of an upper level activity



The teacher gives a short introduction to the students about the tasks to perform and how to handle the different environments

Each group has to present the current results. After that, all developed mazes and rule sets are accessable for all groups.

The maze devolopers first have to install the software on the robot

When one of the subgroups requests it, a discussion about the strategies

The strategy devolopers are working with the pc because it is possible to work much faster in the simulation than with the robot.

Figure 3: The user interface of the diagram-based authoring tool

Figure 3 shows two screenshots of the maze script representation (control flow) in the diagram based modelling tool. On the left, it shows a fragment of the upper level control flow of activities. Here this is a simple sequence. The designer can decide for each activity node to show either a pedagogical comment (like e.g. the upper activity) or a sketch of its sub activities (like the activity node in the middle). She can also assign and denominate roles, tools, etc. to an activity which can be used when defining sub activity flow. This can be done by double clicking on an activity and can be repeated to any level. A new workspace is opened like it is shown on the right part of Figure 3 which is a screenshot of the whole tool. On the right border you can see the drag and drop area of the tool. This tool is realised as a plugin for our Cool Modes modelling environment (Pinkwart, 2003).

POTENTIAL USES AND SYSTEM SUPPORT OF CSCL SCRIPTS

The specification of learning processes using a modelling language may have a broad variety of purposes on the designer’s side. Some educational designers use it as a note taking tool for for lesson planning, some for discussion with colleagues and some expect these models to be executed automatically within a customised computer-based learning environment. With this in mind, we want to explore and elaborate the different motivations designers might have and potential functionalities a run-time system may provide from a given learning process specification. One of the dimensions for our exploration is the degree of “informedness” on the part of the computer system necessary to provide the desired functionality: we see a continuum from complete uninformedness of the system and exclusive interpretation on the user’s side up to quite high requirements of interpreting/understanding the learning processes within the system. We will begin our discussion on the end of the “uninformed system”.

System as Editor/Viewer:

Basically, designers of learning processes can use their models as blueprints for their teaching or experimental design. When the model is also used to communicate the process to other actors (other designers, experimentors, students) the modelling language has the function of a communication language, the learning process models have the function of a shared artefact in the communication. A well-known example for this type of usage of modelling languages is the Unified Modelling Language (UML) as a lingua franca for software design and specification. We think that a visual modelling language for learning processes with an elaborated conceptual model could be of similar importance to learning designers as UML is for software engineers. This type of usage does not require any interpretation on the part of the system, but nevertheless the system can provide the syntactic elements of the modelling language in form of a model editor with persistent storage and thus re-use and exchange of models. When visualising the learning flow to the learners directly, this artefact can be seen as a navigation structure and therefore a means of self-reflection on the learner’s side (“What should I do here?”), even without any monitoring capabilities of the system (we will go into detail on that at the end of the section).

Syntactical mapping to a visual/conceptual representation:

If the system has an explicit representation of the syntactical elements not only on the user interface level but also in the internal data model, the system can additionally provide a mapping from syntactically defined learning processes to a user-understandable visual representation. This enables interoperability in terms of the exchange of learning process specifications that have been produced with different tools given a compatible syntax (either the same or transformable via a syntactic mapping). With a well-defined syntax available in the system, tools can support the learning designers in producing syntactically correct models (with techniques like highlighting of syntax errors) an important prerequisite for further processing of the model.

Presentation of models in multiple perspectives:

Rich, expressive modelling techniques usually bring along the problem that models get excessively complex and hard to overlook. Therefore either reduction of the complexity (by applying projections of specific elements or filtering techniques) or the separation into different perspectives (like the different diagram types in UML) is a typical way to cope with the complexity. For learning processes typically the following aspects are relevant and thus candidates for special perspectives:

- *Procedural/Temporal Perspective*: naturally the sequence and timing, i.e. the processual aspects of the whole learning process should be represented explicitly
- *Artefacts Perspective*: artefacts given as resources, used as temporary results and the final outcome of learning activities constitute an important aspect of learning processes. Especially the change of artefacts over time (version history) is information to consider by all participants of a learning process.
- *Roles Perspective*: for organisation of specific tasks in group processes the various roles needed for the tasks are an essential information both for designer and for learners. With a suitable perspective the designer keeps the overview about the work organisation and the learners can reduce their uncertainty (Mäkitalo et al. 2004) about their role, i.e. the function that is expected from them, in the learning process.
- *Individual/Group Perspective*: to get an impression of the workload of one specific member or one subgroup within a group process a perspective stressing these individual aspects is a valuable information for the designer to keep balance between the participants of the process. For the participants this perspective can give orientation about their progress and a ‘ToDo list’ as a scaffold for their activities.

The multi-perspective representation of the learning design requires that the system explicitly has information which elements belong to which perspective(s), especially when relations between perspectives should be highlighted. Therefore an additional level of information, i.e. the assignment of syntactical elements to the different perspectives has to be present to support the users properly.

Up to this point, we can call the computer support described so far as a rather syntactic, without considering higher-level properties of the learning flow, such as executabilty or structural aspects on semantical level. In the following paragraphs we will shed some light on this kind of advanced support mechanisms:

Model-based prediction

An explicit representation of the model can be used to give advise or comments to the designer of the learning process with respect to her design: E.g., dependencies or constraints between elements can be highlighted, such as necessity of sequential phases or synchronising the flow after a split into cooperative subprocesses. If the designer specified temporal constraints (minimum or maximum time) for elements of the process, techniques from operations research, such as optimisation in network flows or critical path analysis can be applied. With this kind of support the designer can find weak spots in the design, such as an inappropriately long waiting time for the participants in one subprocess when synchronising with another subgroup whose activties take much more time. Scheduling algorithms may propose a different sequencing for the revision of the design then.

Simulation

A simulated execution of the specified learning process can give the designer a more profound feedback on "what works and what does not?". Imagine the benefit of doing a "simulation run" with information about sequence, time requirements, produced artefacts before applying the whole design to a real experiment. The plausibility of the design can be checked much easier than just based on the static structure of the model. This clearly requires an "operational semantics" of the learning process modelling language to provide execution runs of the specified learning process (e.g. operationalising which activity follows which others and how to complete/end activities). It should be possible to explore such a simulation interactively and stepwise, for more thorough testing of the processes' feasibility a "batch" mode or even exhaustive simulation with different inputs may be desirable. Deadlocks (e.g. when subgroups are waiting for each other's input) in the process specification can be detected before making the bitter experience in practical use. The degree of detail for simulation will also vary here, from rather general level, such as interactively giving a specific ordering of activities, to full simulation of the users' interfaces which would be equivalent to a full-fledged execution engine for the process and thus to the expected functionality of a "player".

Static configuration of the learning environment

The first, weak approach to operationalising the learning process for the target user "at run time" is the configuration of the learning environment with available tools, resources, communication structure and so on. If this configuration is done once without dynamic addition and removal of elements we call this static configuration. This gives the target users the full potential of available elements, but without the constraints and restrictions that may have already been specified by the designer. "Compiling and instantiating" such an environment from the specification should be the minimal functionality of a system meant for "playing" the learning design.

Monitoring of the learning flow

Enriched by computable conditions how and when to end activities, how to measure the progress state of artefacts etc. the operational character of the computer support can be substantially enhanced. Given this additional information, monitoring of the learning flow and management of the constraints specified by the designer is possible. The computer support takes the form of a fully operational execution of the process. This level has not yet been achieved for IMS/LD players, because Level A compliance is not sufficient to handle complex conditions that influence the flow at runtime. Monitoring functionality could be used twofold: On the one hand the information can be used internally to adapt the process according to the exact specification, on the other hand the monitored information can be visualised to participants of the learning process and give them information on what they have done and produced. This additional feedback can be used to promote reflection about the process or the participants' own behaviour, such stimulating meta-cognitive activities. Yet, just thinking of execution of a predefined learning process is not enough: Modern interactive learning environments allow the learners to structure their learning process very flexibly. Here, the recognition problem is known to be very hard. On the other hand, the strive for learning monitors should not induce additional restrictions on the learner.

Model-based scaffolding

At the "informed end" of the spectrum of computer support we see the potential use of the system for scaffolding the learning process, especially when the "typical path" through the process was left by the participants. An enriched specification can give advise to the learners on "what and when to do, how they can play their assigned role best" and so on. Depending on the strictness of the scaffolding the system's behaviour can vary between an unrestraining advisor and an interventing tutor. For this functionality, existing approaches both from the area of "intelligent tutoring systems" and from "interaction scripts" as discussed in cognitive and social psychology can be considered. The information needed for this functionality has aspects of operational character (what to do in special circumstances) but also of the "rationale" the designer had in mind with the specified activities (why is that activity important and how to implement it).

CONCLUSIONS AND FUTURE WORK

In this paper we have stated five major limitations of IMS LD when formalising CSCL scripts. Based on this, we have suggested a scripting language for CSCL. The identified problems of IMS LD are solved in the language respectively by: 1) explicitly introducing the group entity to facilitate modelling organisational role and behaviour role; 2) explicitly introducing the artefact entity to enable designers to model artefact and information flow easily and intuitively; 3) extending process element operations and providing declaration mechanisms to capture dynamic features of collaborative learning processes; 4) exploiting WfMS routing technologies to enable specifying complicated control flow; and 5) giving up the metaphor of theatrical play and the role-part and using activity-centered definition method to model varied forms of social interaction. Furthermore, we systematically discussed the potential usages of CSCL scripts and possibilities of system support.

Based on the CSCL scripting language, we developed two script authoring tools using different representation perspectives: the tree-based view and the diagram-based view. Currently we focused on testing the modelling capacity of the proposed CSCL scripting language, in order to improve the capacity of our language and the feasibility of the tools. Real experiments will be conducted in the near future. Later on, we will develop an integrated environment to provide a full spectrum of system support from modelling, analysing, simulating, monitoring, to scaffolding.

ACKNOWLEDGEMENTS

This research has been supported by grants of the European Commission for the Kaleidoscope Project (IST-2004-507838).

REFERENCES

Caeiro, M., Anido, L. & Llamas, M. (2003). A Critical Analysis of IMS Learning Design. In Proceedings of CSCL 2003, p.363-367.

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 (p. 61-91). Heerlen, Open Universiteit Nederland.

Guzdial, M., Turns, J. (2000). Effective discussion through a computer-mediated anchored forum. Journal of the Learning Sciences, 9(4), p437-469.

Hernandez, D., Asensio, J.I., Dimitriadis, Y. (2004). IMS Learning Design Support for the Formalisation of Collaborative Learning Flow Patterns. Proceedings of the 4th International Conference on Advanced Learning Technologies (Aug.30 - Sep. 1, 2004, Joensuu, Finland), IEEE Press, pp.350-354.

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.

IMS LD home page: http://www.imsglobal.org/learningdesign/index.cfm (download on 15/09/2004)

Jansen, M., Oelinger, M., Hoeksema, K., Hoppe, U. (2004). An Interactive Maze Scenario with Physical Robots and Other Smart Devices. In: Proceedings of the 2nd IEEE Intl. Workshop on Wireless and Mobile Technologies in Education, WMTE 2004, Los Alamitos, California (USA), pp 83-90

Kollar, I., Fischer, F., & Hesse, F. W. (2003). Cooperation Scripts for Computer-Supported Collaborative Learning. In B. Wasson, R. Baggetun, U.Hoppe, & S. Ludvigsen (Eds.), In: Proc. of CSCL 2003, COMMUNITY EVENTS - Communication and Interaction (pp. 59-61). Bergen, NO: InterMedia.

Koper, E.J.R. (2001). Modeling Units of Study from a Padagogical Perspective: the Pedagogical Meta-model behind EML. Educational Technology Expertise Centre Open University of the Netherlands.

Koper, E.J.R. & Olivier. B. (2004). Representing the Learning Design of Units of Learning. Educational Technology & Society. 7(3), p.97-111.

Mäkitalo, K. , Weinberger, A. , Häkkinen, P., & Fischer, F. (2004). Uncertainty-reducing cooperation scripts in online learning environments. In P. Gerjets, P. A. Kirschner, J. Elen & R. Joiner (Eds.), Proc. of first joint meeting of the EARLI SIGs "Instructional Design" and "Learning and Instruction with Computers"

Miao, Y., Holst, S., Haake, J.M., and Steinmetz, R. (2000). PBL-protocols: Guiding and Controlling Problem Based Learning Processes in Virtual Learning Environments. In: *Proceedings of the Fourth International Conference on the Learning Sciences (ICLS'2000)*, pp. 232-237. June 14-17, 2000, Ann Arbor, U.S.A.

MOSIL home page: http://www.craftsrv1.epfl.ch/mosil (download on 20/11/2004)

O'Donnell, A. M., & Dansereau, D. F. (1992) Scripted Cooperation in Student Dyada: A Method for Analyzing and Enhancing Academic Learning and Performance. In R. Hertz-Lazarowitz and N. Miller (Eds.), Interaction in Cooperative Groups: The theoretical Anatomy of Group Learning (pp. 120-141). London: Cambridge University Press.

Pfister, H.-R. & Mühlpfordt, M. (2002). Supporting discourse in a synchronous learning environment: The learning protocol approach. In proceedings of the CSCL2002, p.581-589.

Pinkwart, N. (2003). A plug-in architecture for graph based collaborative modeling systems. Proceedings of the 11th Conference on Artificial Intelligence in Education, Sydney, Australia.

Schank,R.C. & Abelson, R.P. (1977). Scripts, plans, goals and understanding. Hillsdale, NJ: Erlbaum.

WfMC homepage: http://www.wfmc.org/index.html (download on 20/11/2004)