*Journal of Universal Computer Science, vol. 14, no. 1 (2008), 148-173*
*submitted: 6/11/06, accepted: 17/10/07, appeared: 1/1/08* 

# The Remote Control Approach - An Architecture for Adaptive Scripting across Collaborative Learning Environments

Andreas Harrer, Nils Malzahn, Astrid Wichmann
University of Duisburg-Essen, Germany
{harrer, malzahn, wichmann}@collide.info

**Abstract:** In this article we present an architecture for the integration of tutoring approaches and process scaffolds into existing collaborative applications. The architecture allows to combine existing research on explicit representations of collaborative learning processes with the availability of existing and tested collaborative learning environments. The architecture allows to control the learning environments either by a human or a pedagogic agent and thus enables adaptation of the tools to the current state of the learning process. Both types of tutors are using a so-called remote control component using the same set of control primitives. To prove the soundness of the architecture and the flexibility of its implementation two example scenarios are shown that use IMS LD learning process definitions with the Coppercore learning design engine controlling our collaborative environments.

**Key Words:** Collaboration Scripts, Integration Architecture, Learning Design
**Category:** D.2.11, K.3.1, L.2.0, L.6.2

## 1 Introduction – Structuring and Scaffolding Collaboration

Collaboration has become an important factor in learning activities, especially in disciplines that require substantial phases of working in teams, such as computer science, communication sciences etc. This can be seen in the emergence of the research field Computer-Supported Collaborative Learning (CSCL) in the last decade. Yet, just reducing the computer-based support to the provision of the suitable technological means to communicate, is most often not sufficient to promote the collaborative learning activity. Studies like Weinberger [Weinberger 2003] showed that collaboration does not happen effectively in every situation just by initiating the collaborative situation.

The two approaches of scaffolds [Quintana et al. 2002] respectively collaboration scripts [O'Donnell et al. 1992] are means to structure the learning activity and to support the learners in organizing their activities or acquiring the skills to collaborate effectively. Thus their use in computer-based learning support environments (LSE) is a major topic of recent research in the CSCL community ([Weinberger et al. 2005a]; [Harrer et al. 2005]). At the moment the term "script" is used in a highly ambiguous way: The pedagogical rationale of a collaborative learning activity, that shapes the general context and sequence of the whole activity is called a CSCL script. An example for this is the stimulation of collaboration by splitting the task so that interaction happens at the split (cf. [Dillenbourg 2007]). Yet the same term is used as well for the fine-grained prescription how argumentation should happen in complete argumentative sequences (cf. [Leitao 2000]).

Interestingly a parallel discussion occurs also in a field of computer-supported learning that has evolved independently of CSCL, the discipline of Intelligent Tutoring Systems (ITS): The support of the learners by the system to promote them in the learning process is often called tutoring or interventions. The components called "intelligent tutors" or "pedagogical agents" are used comparably ambiguous as the term "script" in CSCL with respect to the granularity and competencies the tutor/agent should provide.

It is obvious that the expertise and experiences of these two fields should be combined in collaborative computer-supported learning activities. One of the grand challenges for the shared interest between the communities will be the representation and implementation of scaffolds respectively tutoring processes for collaborative scenarios. The definition of formal models for collaboration support results in the explication of the pedagogical and psychological rationale for the scientific exchange between researchers and practitioners, but also in the practical application of the models in computer-based learning environments.

This article will present our approach of combining aspects from CSCL, pedagogical design, and ITS in an integrated architecture to support collaborative learning activities. For that end we discuss in section 2 the desirability of explicit and formal models for learning processes to make the pedagogical rationale that is represented within these models re-usable more efficiently. Section 3 presents our proposal for the combination of existing collaborative applications with engines for the execution of learning processes conceptually and in the following a flexible architectural design for this approach. The implementation of this approach is described in section 4 using the third-party learning design engine CopperCore and our collaborative modelling applications Cool Modes and FreeStyler. In section 5 we show the expressiveness and scope of potential usages for learning scenarios by the description of a complex adaptive scenario for scientific inquiry learning and a collaborative scenario involving variation based on the result of a group decision. Several challenges for the research field and our approach with respect to the usage of heterogeneous learning tools and the re-use of learning products are discussed in section 6, before our conclusions and future directions are presented in section 7.

## 2 Formal Models of Learning Processes and Collaborative Applications

Up to now complex learning support environments and explicit scaffolding/tutoring models are largely unrelated and co-exist, but do not co-operate. On the one hand LSEs, such as WISE [WISE], CoLab [van Joolingen et al. 2004] or Belvedere [Suthers et al. 1995], either have a specific ("hard-wired") process model embedded or do not have an explicit learning process model at all. On the other hand environments that use explicit process models for supporting the learning process, typically fall short in at least one of these two criteria:

1. Re-usability of the process model in other contexts: most systems using a formal model for structuring the interaction between the learner(s) and the system define their own proprietary model for the learning process which is not understandable and thus re-usable by other applications: this may happen because of proprietary formats that cannot be mapped to other formal approaches, a lack of explicitness of the operational semantics of the model, or a lack of explicitness of the model itself, which is often deeply intertwined with the graphical user interface. Among the explicit models for defining the learning process are production rule systems [Aleven et al. 2004], automata-based [Martens 2004] and flow-oriented models [Dawabi and Wessner 2004].

2. Expressiveness for complex learning processes: systems that have explicit mechanisms for structuring activities usually tend to have a very narrow focus, such as sequencing the presentation of learning material in web-based hypertext systems or intervening on the first deviation from an "ideal" learning path [Anderson et al. 1995]. Especially more coarse-grained learning activities, such as experimentation, model construction, and argumentation are usually not scaffolded in these systems. There are very few approaches, that explicitly try to scaffold collaboration with adaptive approaches: coming from the ITS area, the "Collaborative Tutor" approach [McLaren et al. 2005] attempts to support the fine grained level of user actions in relatively small problem-oriented tasks, such as object-oriented modelling. The GridCOLE approach [Bote-Lorenzo et al. 2004], that is rooted in the CSCL field, combines the explicit description of coarse-grained learning activities with the launching of services suitable for the specific activity. IMS Learning Design [Koper and Tattersall 2005] (IMS LD or short LD) can be considered a formal approach with explicit representation of both the models and the operational semantics, even though both aspects could be discussed even more precisely as proposed in [Amorim et al. 2005]. Comparable approaches for modelling of learning scenarios are the Learning Activity Management System [Dalziel 2006] (LAMS) that allows definition and execution of learning processes by a drag-and-drop definition of processes via a set of defined activity tools or the Learning Design Language [Martel et al. 2006] (LDL) that uses a graphical notation to specify learning processes and automatically execute them in a built-in runtime mode. Surprisingly up to now learning design documents as process scaffolds or "scripts" are usually oriented towards delivery of web-content and some simple services, such as conference

tools. Yet, making the learning processes explicit in a formal specification, such as IMS LD, offers also the possibility to re-use the pedagogical rationale that is reflected within the specification and to define more complex learning activities than just sequenced content delivery.

We also assume that the formal character of IMS LD or similar languages can be utilized to scaffold and apply tutoring support for pre-existing LSEs. The availability of learning design engines (LDE), such as CopperCore[1] or the Learning Design Infrastructure (LDI) associated with the LDL approach, can provide explicit process support without having to implement a process model from scratch for each individual environment, if we can meet the challenge of integrating pre-existing LSEs and LDEs in a flexible, interoperable architecture.

In the next section we will present our approach to achieve this synergy between the both lines of computer-based learning CSCL and ITS manifested in an architecture supporting this approach. In the subsequent section we will discuss both the re-use of pre-existing learning support environments and the re-use of artefacts within a learning process in an implementation utilizing the IMS-LD standard for learning processes and an IMS-LD engine for the execution of complex learning processes in external learning support environments.

## 3 A Flexible Architecture for Tutoring in Collaborative Settings

We propose an approach that aims at a clear separation of the learning design engine together with the specification and implementation of the learning flow (as LD documents) and the collaborative learning environments. In this proposal we assume that the learners interact exclusively with the LSE without having to know anything about being "scripted" or "scaffolded" by the LDE respectively the LD document. According to Vogten, Koper, Martens, and Tattersall [Vogten et al. 2005] learning design engines can be represented as a collection of finite state machines that react to changes of properties with state transitions by sending events of a specific output alphabet. In the loosely-coupled connection of an engine with a learning support environment presented in figure 1, the engine controls the learning environment with output events (such as "start a new phase", event 1.), defined as a vocabulary for a set of environments, that are mapped by the environment to its existing functionality (such as "create new workspace", the configuration of the LSE through event 1.1).

Since the LDE interacts closely with the LSE, the LSE is – in relation to the IMS LD proposal – more than an IMS LD service which is not monitored and controlled by the LDE during the activity. The learners interacting with

[1] CopperCore — The IMS Learning Design Engine, `http://coppercore.sourceforge.net`

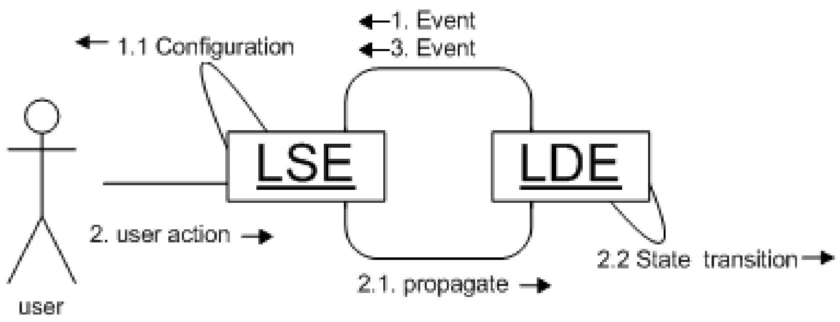


Figure 1: UML communication diagram for interaction schema between LDE and LSE

the learning support environment create events (user action 2.), such as "phase is completed" (either directly or monitored by the LSE), that map to the input alphabet of the engine's state machines and are propagated to the LDE (message 2.1). The triggered state transition (message 2.2) causes the learning process to advance and will again trigger control messages (event 3.) to be accepted by the LSE. In that way we get the regulation cycle of figure 1 with the LDE and the LSE influencing each other's state. Using a generic vocabulary of communication primitives between the LDE and LSE has the advantage, that the learning process model can be used with a variety of different LSEs without any changes to the document, given that the LSE can make use of primitives of the vocabulary.

For the concrete realization of our approach we defined an architecture that brings together LSEs and LDEs without having to make substantial changes in either of the two components: This is achieved by introducing an intermediary generic component, called *remote control*, that loosely couples the LSE and LDE by means of abstracting from specific dialects of the learning design and of the learning support environments using a defined vocabulary (see below). The schematic overview of the architecture can be found in figure 2 and the components introduced have the following function:

**Engine Extension (CopperCore Extension):** this component extends the event propagation mechanism of the learning design engine, so that on state transitions within the engine, events are sent to the LSE to remotely control the learning process according to the LD document's description. This event is sent indirectly to the LSE via the Remote Control Component to provide less coupling and more flexibility between LSEs and LDEs (see below). For the concrete realization of the architecture we aimed to use a third-party LDE that uses both an expressive learning design dialect and also provides

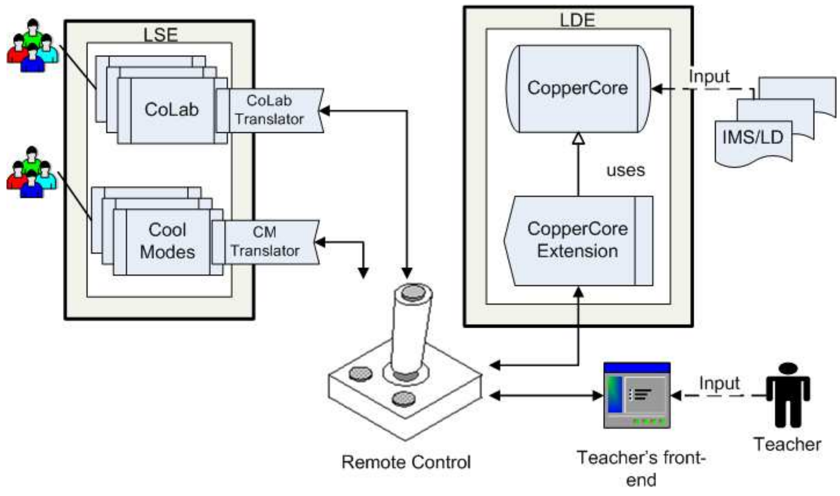


**Figure 2:** Remote Control Architecture for interaction between LDE and LSE

access to its interfaces and / or source code to allow for the extensions necessary to our approach. The CopperCore engine is an open source LDE for the IMS LD language, which satisfies the availability criterion and the expressiveness criterion to a wide extent [Hernandez-Leo et al. 2005]. Alternatives, such as the LDI / LDL combination are not freely available, or in the case of LAMS require substantial changes for the learning tools to be integrated as new activity tools. Thus our technical approach is based on the use of a modified CopperCore engine.

**Remote Control Component:** this component is the mediator between LDE and LSE; it maps events coming as messages of the specific LD dialect from the LDE into one or more communication primitives, that build the vocabulary for remotely controlling learning support environments, such as CoLab, FreeStyler, or Cool Modes [Pinkwart 2005]. Using this vocabulary allows for a flexible exchange of both LSE and LDE, because their communication is not tied to specific dialects of learning flow specifications, i.e. the LSE does not have to understand a specific learning process specification. These abstract "commands" are then sent to the "remote API" of the specific LSE.

**LSE Remote API (Translator):** this interface accepts communication primitives that have been defined for a variety of different LSEs and maps these primitives to the specific functionality available in the concrete LSE. For example the communication primitive "Show workspace for voting phase" could be mapped to calling the functionality "Make visible a workspace with title 'Decision on Solution' and add a Voting Plugin" in the Cool Modes environment (see figure 4). The primitive that has been sent out from the Remote Control Component to the subscribers of this primitive (all LSEs that understand the primitive) is then translated to a call of the respective functionality of the LSE; thus this can be considered a remote call of the LSE functionality by the Remote Control Component. Similarly, actions taking place in the LSE that are relevant for the LDE to monitor the state of the learning process, are propagated be the Translator so that the LDE can update its state accordingly (message 2.1 in figure 3). To integrate an LSE into the proposed architecture, a translator mapping the command primitives to the LSE in question and to propagated LSE events has to be implemented, i.e. there is a one-to-one relation between an LSE and a Translator.

An interesting feature of this architecture is, that besides our main purpose, i.e. the realization of collaboration scaffolds in pre-existing learning support environments, the remote control can be used by a variety of different actors (in the socio-technical sense of actors being both humans and technical systems):

- A virtual agent/tutor, that has some model for scaffolding/tutoring the learning, such as in [Martens and Uhrmacher 2004], if it uses messages that can be mapped by the remote control to the communication primitives of LSEs. The LDE component can be considered our standardized type of such a virtual agent using the IMS LD dialect, but it could be replaced by a full-fledged intelligent tutoring agent regulating the learning process adaptively using the remote control vocabulary. Other types of tutors, such as the Cognitive Tutors [Anderson et al. 1995], would then be usable, given that they comply to a design similar to the tool-and-tutor architecture discussed in [Ritter and Koedinger 1996].

- A human teacher, who can react at runtime to the learning situation and give hints, additional tools, and / or instructions, as she or he thinks are appropriate. This intended use-case of the remote control can be seen in the lower right of figure 2, where the teacher has a dedicated user-interface. This is especially useful in ill-defined domains, such as argumentation, law, and other domains that are hardly analyzable automatically [Aleven et al. 2006]. Here it is possible to let a teacher participate in the learning activity as a moderator like in the Argunaut project[2], a recent EU-funded research and development project aiming at the support of moderators of electronic discussions.

- A human administrator, who can use the remote control with a similar user interface to the teacher's to setup collaborative sessions, assign rights and

[2] `www.argunaut.org`

roles to the students. Such an interface can substantially reduce the administrative effort in setting up experiments and practical use of learning support environments, as was prototypically shown in an early version of our architecture in [Kuhn et al. 2005].

Of course the remote control can be used by several actors at the same time (e.g. a teacher and a virtual agent), but this might produce inconsistencies in the regulation of the process by the agent. This is the case when the teacher intervenes into the collaboration without the agent being able to interpret this intervention, thus reaching an undefined or incorrect state. On the other hand this provides a convenient way for the teacher to react to unforeseen situations in the learning activity, such as breakdowns of groups or tools, where an agent would not be able to help anyway. In the next section we will present some details of our concrete prototypical implementation of the Remote Control Approach and architecture.

## 4 Prototypical Implementation

As a proof of our concept we chose to combine the collaborative applications Cool Modes and FreeStyler with the CopperCore Engine, currently the most advanced IMS LD engine and the Reload LD Player[3], a graphical interface for run-time configuration of learning designs. Cool Modes [Pinkwart 2005] and FreeStyler are applications that allow graphical modelling and handwriting in individual and collaborative use. Among the graphical models used are formal representations, such as finite automata, system dynamics [Forrester 1968], and also flexibly usable representations, e.g. for mindmapping, brainstorming etc. Examples for the usage of these applications in conjunction with our scripting approach can be found in section 5. CopperCore and the Reload Player both are third-party applications with well-defined interfaces, so they can be adapted to the specifics of our proposed approach.

The prototypical implementation of our architecture will be described in three steps: In the first step we will describe the extensions of the CopperCore engine, in the second step we will sketch the implementation of the Remote Control plus an extension of the Reload IMS LD Player as a teacher's frontend and in the third step the extension of the Cool Modes translator (cf. figure 2) will be explained.

### 4.1 CopperCore

Since the CopperCore engine shall be used as an agent that uses the Remote Control, it is essential to be able to receive events from the engine. This is done

[3] RELOAD Project – Learning Design Player `http://www.reload.ac.uk/ldplayer.html`

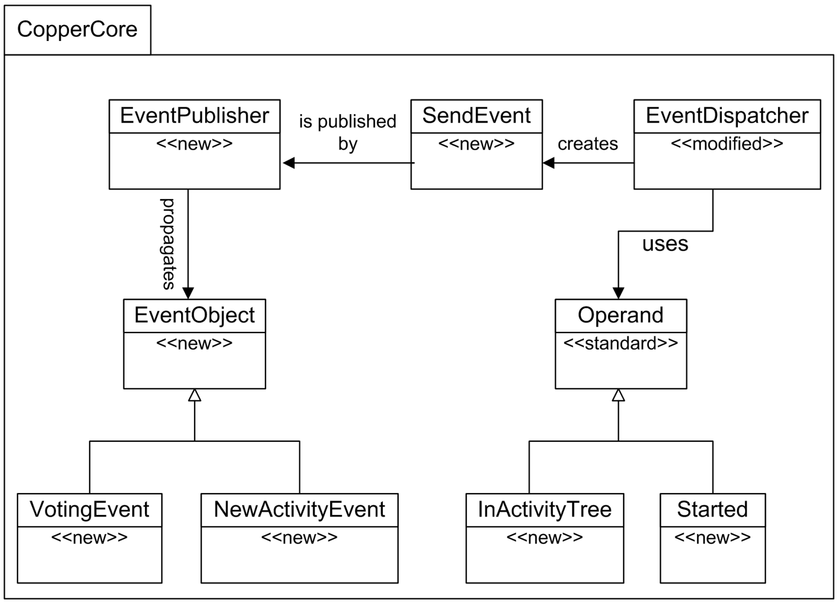


**Figure 3:** UML-like diagram for extensions of the CopperCore engine

by adding an *EventPublisher* to the CopperCore engine that notifies the Remote Control when changes of CopperCore's inner state machine occur. Another possible approach would have been to poll the CopperCore's inner state continuously. Although this approach may have avoided changes to the CopperCore engine completely, we decided to implement the first option to avoid the resource consuming polling requests of Remote Control Component. CopperCore already provides an event propagation mechanism managed by an *EventDispatcher* which is used internally to calculate the consequences of advancements in the Learning Design script (e.g. if a learning-activity is completed, a role-part is completed). The given set of expressions and actions of the original CopperCore engine were extended by introducing the conditions *Started* and *InActivityTree* as well as a class called *SendEvent* that notifies the Remote Control Component via the EventPublisher. These two new conditions help us to synchronize the Remote Control with the CopperCore engine and to distinguish which collaborative application has to be contacted because of the particular change in the activity tree.

The parser of CopperCore has been extended to fire the SendEvent whenever an activity ends. This was necessary to ensure that the SendEvent is executed if the state changes are of interest (e.g. if an act has been made visible). Whenever a SendEvent occurs, an appropriate EventObject (e.g. a NewActivityEvent) is created and sent to the Remote Control (see figure 3). The communication between the CopperCore engine and the Remote Control component, as well as the communication to and from the LSE, is realized via Java Message Service[4] (JMS), because it was convenient for the presented combination of LSE and LDE. Nevertheless it is possible or even necessary, depending on the controlled LSEs, to exchange the communication channel with other techniques in the future. For the communication format we chose XML, since IMS Learning Design is specified in XML and CopperCore makes already extensive use of it. If an EventObject is created all relevant data is collected from the CopperCore engine and an XML string is constructed. This string is sent by the EventPublisher via a JMS TextMessage to the Remote Control. If a message arrives at the Remote Control the EventObject is re-constructed by parsing the received XML string. EventObject is used here as a placeholder for a family of concrete events like the above mentioned NewActivityEvent. These events are subclasses of the EventObject as shown in figure 3.

### 4.2 Remote Control

The Remote Control is the intermediate device between the learning process management and the collaborative applications. It should be able to run either without human interaction, monitored and operated by a pedagogic (computer) agent or operated by a human. Both types of actors shall be enabled to interact with the learners through the connected LSEs. Concerning the idea of using IMS LD documents to scaffold the learning process we wanted to provide an intuitive and easy-to-use (teacher) interface to configure the LDE (i.e. manage learning scripts, create users, assign them as appropriate etc.). Since the Reload LD Player [Reload] already has a quite intuitive UI and supports to start the CopperCore server we used this application as a starting point to add some functionality. First of all the user management has to combine the internal users of the CopperCore engine and the users sitting in front of the LSEs. The Remote Control maps the Copper Core's participants to the users of the learning environments. In our example the information about the LSE users is provided by the MatchMaker [Jansen 2003] server that is used as a collaboration server for the Cool Modes and FreeStyler environments.

The assignment of users to units of learning can either be done by humans like it was already possible in the original Reload LD Player or by an agent which notices new users and assigns them automatically to a script and starts it afterwards. The human actor on the one hand has the option to create users and assign them from the beginning (e.g. while preparing a lecture) and on the other hand he can wait for the users to start their LSEs, so they will automatically show up in the Remote Control and assign them to a specific script on the fly.

[4] Java Message Service (JMS) `http://java.sun.com/products/jms`

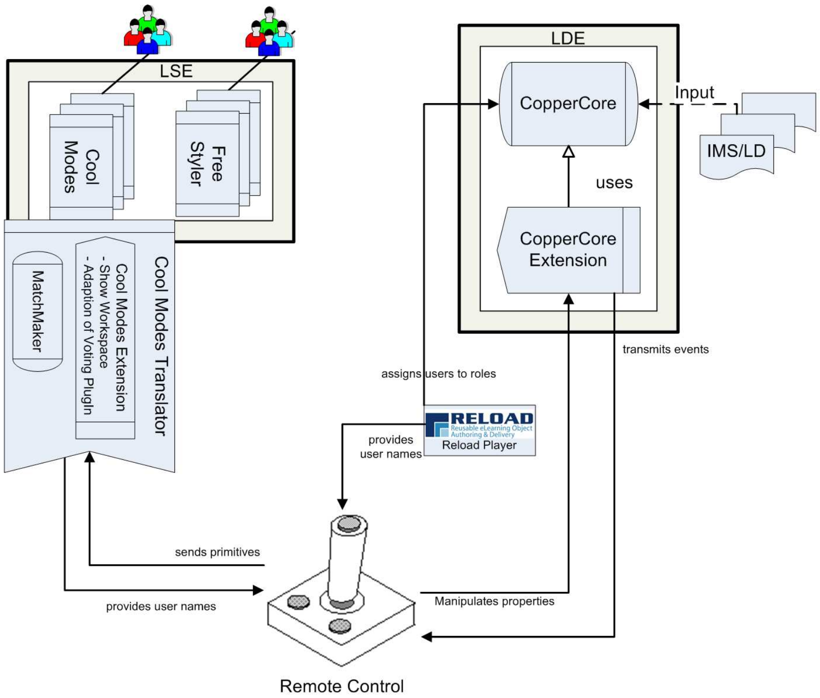


**Figure 4:** Overview of the Remote Control implementation using Cool Modes

As for the communication primitives, that are created from events in the LDE and then sent to the LSE, we started out with a small set of events described above, such as "New Activity", "Voting", and "New Tool". A similar approach of mapping tools to learning activities that uses ontologies for the mapping has been developed in the OntoolCole project [Vega-Gorgojo et al. 2006]. After a review of literature in the field of collaboration scripts, e.g. [Weinberger et al. 2005b], [Kollar et al. 2005], [Mäkitalo et al. 2005], and tutoring interventions such as [Ritter and Koedinger 1996], we extended this set with primitives for "Prompting", "Text scaffolds", "Awareness information", "Highlighting", and "Pointing". While the initial primitives can be considered as means to shape the learning process at a coarse-grained level (i.e. macro-level CSCL scripts), the latter primitives aim at a rather fine-grained granularity of scaffold and support, which could also be used by tutor agents for immediate feedback interventions (i.e. ITS-typical support). The flexible combination of both types of interventions provides a vast potential to support learning processes at different levels of feedback and scaffolding.

### 4.3 Application specific extensions

As stated above our goal is to support a wide range of existing applications, so we conceptualized a Translator which shall be loosely attached to each LSE. Those classes that "translate" the command primitives sent by the remote control to application-specific events or method calls implement the specified Translator interface. These classes are also responsible to "translate" the application specific results to primitives the remote control can handle. Depending on the specific LSE this can be done with more or less effort. In general we think that applications that support collaborative learning should be easily adaptable to be remotely controlled, because in most cases there already are events to be distributed among other clients of the LSE. To make a first proof of this assumption we created a Translator for the FreeStyler application after the initial Cool Modes translator [Harrer et al. 2006], which was very similar to implement. In our examples the Translator can act as an additional client in the collaborative environment. If it is possible to implement the translator in such a way, the LSEs are not required to know anything about the whole learning process and just react to the instructions of the Remote Control, which are again based on the Learning Design engine. So the Translator behaves like a tutoring agent giving instructions how to adapt the LSE to the learning flow from the perspective of the LSE.

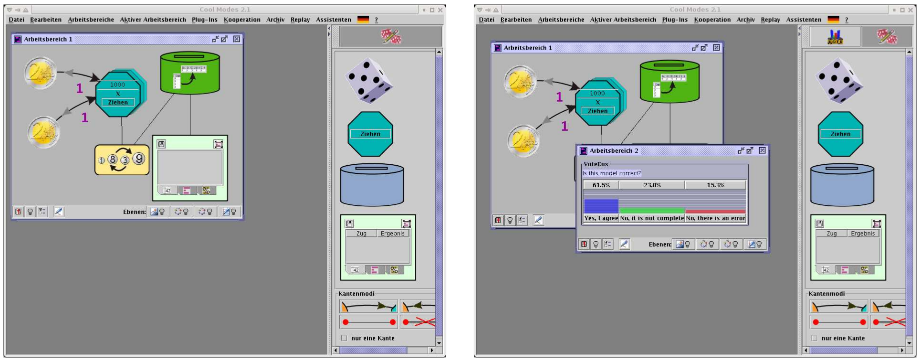

Figure 5: The Cool Modes learning environment before (left) and after (right) transmission of the communication primitive "ShowWorkspace for VotingPhase" from CopperCore Engine. In the right part the voting plugin was added (small icon in top right corner) and an additional window appeared to conduct the voting.

For example during a collaboration session with the goal to model a stochastic experiment (see figure 5) the participants indicate that they do not want to change the model anymore. The users' indication is sent to the CopperCore engine via the Cool Modes translator and the Remote Control. The learning design script changes its states and tells the Remote Control to start a voting activity. In turn the Remote Control distributes a *Show workspace for voting phase* command primitive. The Translator translates this primitive application specific commands for the creation of a new window containing a voting opportunity.

While the availability of a voting option is quite wide spread and therefore reasonable to be taken as a command primitive the concrete implementation of the voting is application specific. So every LSE has to use its own means to enable the students to give their vote. This so called VotingService enables the user to choose between different options which in turn affect the learn flow of the IMS LD play. Technically the voting results returned by the Translators are stored in IMS properties. This enables their use as variables that determine the further steps in Copper Core's learning script. This technique can easily be adopted for other activities like evaluation of tests etc.

All the other communication primitives mentioned in the previous section have also been implemented in our FreeStyler remote API / Translator. We will show the usage of some of them in the following learning scenarios that are supported through our remote control architecture. Since the translator is the sole mediator between the LSE and the LDE, independent evolution of both sides is possible, as long as the mapping of the primitives is valid. New versions of the LSE, e.g. of FreeStyler, can be used without modification, given that the interface of the functionality called by the translator is still preserved. On the other hand new functionality in the LSE, such as a new visual rendering of highlighting objects, can be integrated by small adaptations of the translator that map one of the primitives now to the improved functionality instead of the previously used one.

## 5 Scenarios using the Remote Control Approach

To show the expressiveness of our approach for a variety of very different learning activities, we provide two different scripts in the next subsections: the first one is a complex scientific inquiry script with adaptive feedback, while the second is a collaborative modelling scenario with dynamic flow depending on the students' voting decision. The first scenario emphasizes the possible re-use of scripts with different learning contents by using a learning design engine as a remote controller. This scenario also gives the opportunity to illustrate how the behaviour of the learning environment is adapted according to the input given by the student, i.e. characteristics usually associated with Intelligent Tutoring Systems (ITS). The second scenario emphasizes on the scripted collaboration (CSCL) aspects that are possible using the remote control approach. Both scenarios can easily be adapted to different domains by exchanging the content (i.e.

basically exchanging the content of a referenced text file) of the presented scripts while re-using the structure of the script.

### 5.1 A Scientific Inquiry script conducted with FreeStyler

The Inquiry IMS LD script consists of a macro script integrated with a micro script (cf. [Dillenbourg and Tchounikine 2007]) that determine the workflow and give specific hints respectively. The macro structure guides the student through the experimentation process providing several phases of inquiry [de Jong and van Joolingen 1998], such as the creation of research question, stating a hypothesis, experimentation, analysis etc. The instructor can customize the phases depending on the depth of inquiry that is appropriate. The macro structure of the script is represented visually in the FreeStyler application as one tab resp. page for each phase of the macro script; this process phases can be seen in figure 6 at the top of the window, where the student is currently working on the creation of a Research Question (the active page / tab).

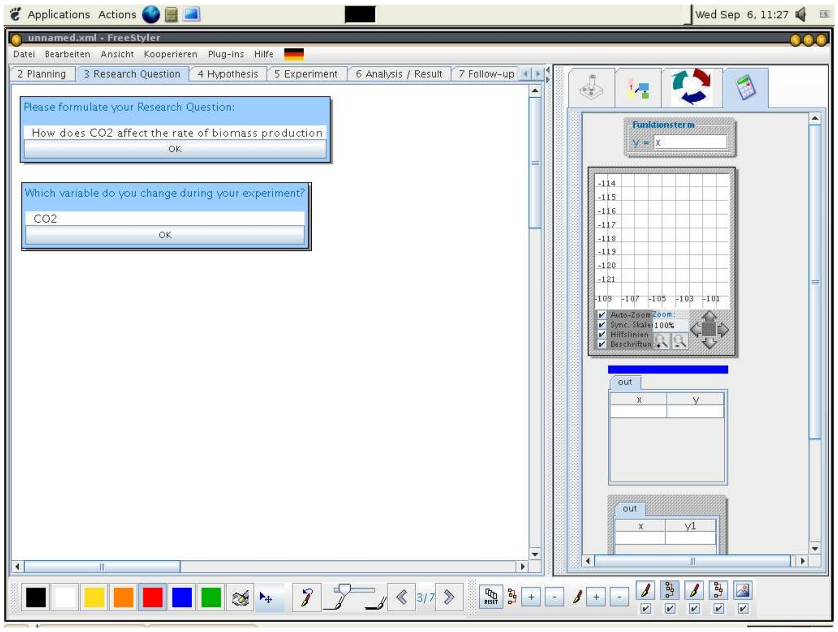


Figure 6: Inquiry Script conducted with FreeStyler - Phase 3 (Research Question)

While this inquiry process is generated as a pre-existing guide or macro-level scaffold, the IMS LD script is also able to generate prompts at run-time that give support during specific tasks of the experimentation process. On this micro level, the instructor can define prompts with means to trigger metacognitive thinking processes that allow the learner to plan, monitor and evaluate their learning process. These micro prompts can be given on different levels of coercion depending on the task difficulty and the level of students' prior knowledge. For example if the students' prior inquiry skills are high because the teacher used authentic approaches to science in class, a less coercive micro inquiry script can be applied. Figure 7 shows a micro prompt with the text "Are you done with the experiment?", that lets the student reflect on the learning process and that also gives informed feedback to the system, when the student commits the prompt.

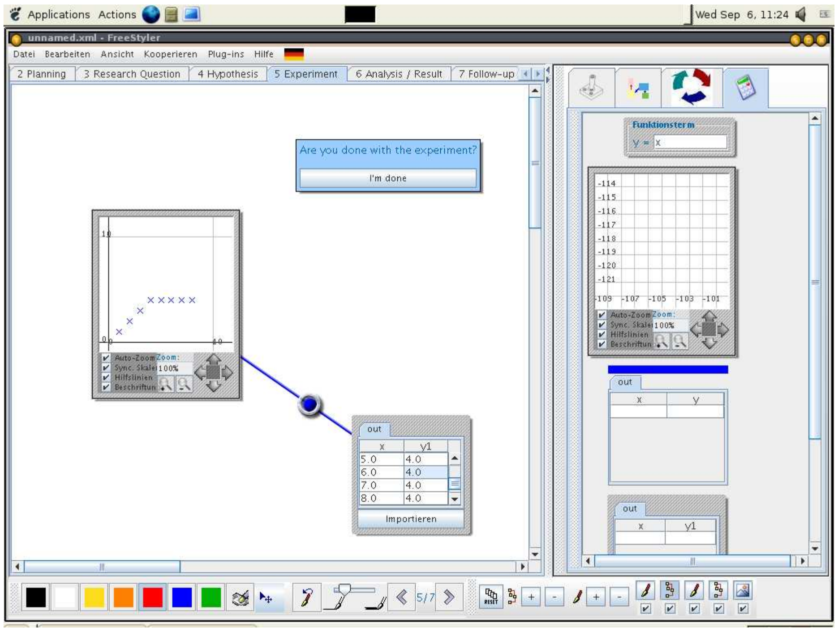


**Figure 7:** Inquiry Script conducted with FreeStyler - Phase 5 (Experiment)

However if more support is needed during the inquiry process, the level of coercion can be adapted. Especially during extensive activities where the student engages in several cycles of inquiry investigating a scientific phenomenon, the micro inquiry script should allow a scaffolding with means of fading the support depending on whether the student just started his or her first experiment or has already finished several experiments. The instructor can employ different types of prompts that can be specified in the IMS LD script:

During our experimentation scenario we used pop-ups to prompt students for monitoring purposes. Pop-ups have a high level of coercion because they force the students to interrupt the activity they are currently engaged in. These monitoring pop-ups remind the students to keep track of mistakes while they are working on a specific problem. An example for such a pop-up can be seen in figure 8 in the lower part.

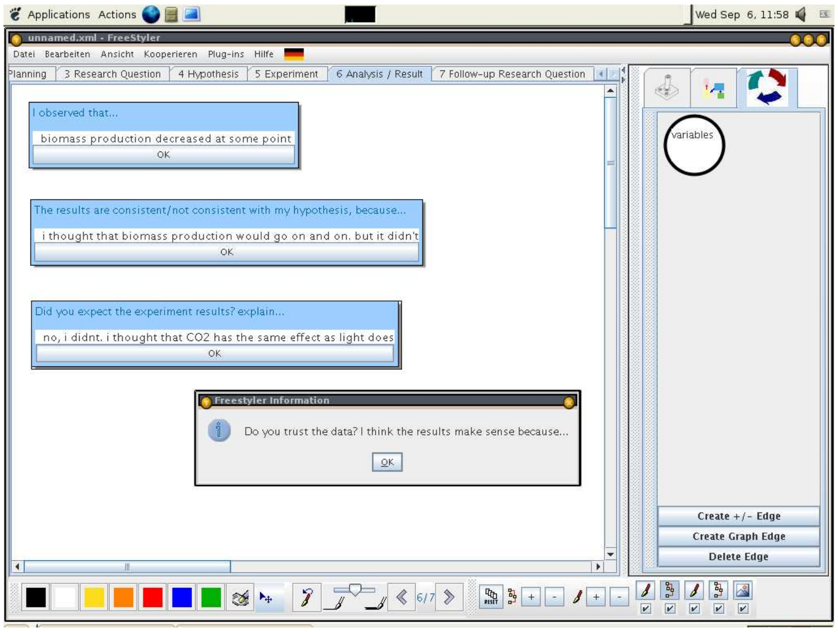


**Figure 8:** Inquiry Script conducted with FreeStyler - Phase 6 (Analysis)

For the planning phase we use embedded hints, which pre-exist on the workspace. These embedded hints are useful for the student to work systematically, such as in figure 6 where the embedded hints scaffold the explication of the research question. Especially when concluding an inquiry phase the students are asked to evaluate and review the progress they have made. Depending on the level of support that is needed we used either pop-ups or embedded hints for supporting evaluative thinking.

The inquiry script is used for students in the age of 15-18 to investigate photosynthetic processes. Students engage in a cycle of inquiry using a simulation program altering parameters that affect plant growth. After a planning phase the students determine conditions of their scientific experiment and generate a hypothesis predicting the outcome. While FreeStyler provides the means of producing a qualitative model and visualising the experiment data using a graph plotter (see figure 7), the IMS LD Inquiry Script structures the learning activity and asks the student to document the experiment throughout the stages of inquiry. For example after students have written their hypothesis they are hinted to explain the reasons of why they think that their hypothesis is correct. During the final stages of the experiment we use pop-ups to ask for specifically reviewing their conclusions with respect to their hypothesis they have made earlier. These prompts aim at fostering the students to externalize their thinking processes to better keep track of inconsistencies that would stay hidden otherwise. We use prompts mainly for the externalization purposes but also to systematize the activity. Leveraging systematic thinking is important especially during our science activities because the problems that students investigate are complex and not easily predictable.

Figure 9 shows the IMS LD content, i.e. an XML document, that is specified by the learning designer to display the topmost embedded hint of figure 6 with the predefined text in the FreeStyler LSE and an associated property that will store the student's statement.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<content>
  <div>
  Please formulate your Research Question:
    <set-property xmlns="http://www.imsglobal.org/xsd/imsld_v1p0"
                  ref="prop-Research_Question"
                  property-of="self"
                  view="value"/>
  </div>
</content>
```

**Figure 9:** IMS LD content representing the embedded hint

The texts of embedded hints can be easily modified by the designer without any programming skills, while the properties usually will have to be defined using IMS LD editing tools. Pop-up messages such as the one in figure 8 can be edited as plain text resource files, so a deeper understanding of IMS LD is not necessary for the use.

The XML declaration of the property used for the student's statement of the research question in the IMS LD manifest can be seen in figure 10. The property value will contain the answer given by the student.

Figure 11 shows the LD fragment from the manifest that defines the GraphPlotter object which is mapped in the Remote Control to a command primitive requiring a plotting tool from the application used in the activity (here the graph plotter of the FreeStyler visible in figure 7 as the visual representation of this LD specification mapped to FreeStyler's functionality).

```
<imsld:locpers-property identifier="prop-Research_Question">
   <imsld:title>Research_Question</imsld:title>
   <imsld:datatype datatype="string" />
</imsld:locpers-property>
```

Figure 10: IMS LD property defined to capture the student answer of an embedded hint

```
<imsld:locpers-property identifier="prop-Experiment_Done">
   <imsld:title>Experiment_Done</imsld:title>
   <imsld:datatype datatype="boolean" />
   <imsld:initial-value>false</imsld:initial-value>
</imsld:locpers-property>
...
<imsld:environment identifier="env-c8d272fe">
  <imsld:title>FunctionGraphPlotterEnvironment</imsld:title>
  <imsld:learning-object identifier="lo-8359f4b0"
                         class="FunctionGraphPlotter"
                         isvisible="true" type="tool-object">
     <imsld:title>FunctionGraphPlotter</imsld:title>
     <imsld:item identifier="item-3b7de949"
                 identifierref="resource-be848c0c"
 isvisible="true" />
  </imsld:learning-object>
</imsld:environment>
```

Figure 11: IMS LD fragment used in the example script - Declaration of a property and a learning environment

The presented Inquiry Script has been recently used with a school class in the field to conduct an experimental study about the effects of fading in scientific inquiry processes. While the scientific evaluation of the study is currently going on, the technical implementation showed quite a robust behaviour and effectiveness in the classroom with a dozen of LDE/LSE systems running in parallel.

### 5.2 A Collaborative Modelling Scenario conducted with Cool Modes

To conclude this section we describe the scenario and the IMS LD script for an exemplary scenario using the collaborative features of the Remote Control Approach and the Cool Modes application. The scenario that has also been shown in figure 5 consists of three phases, the first a collaborative modelling activity of a stochastic mathematical model, followed by a voting activity to decide if all collaborators think that the model is sufficient. If the voting is uniformly positive, the students advance to the presentation of their results (e.g. to their classmates and teacher in a classroom scenario), while otherwise they return to an additional phase of modelling to revise their solution.

The IMS LD document describing this scenario consists of a one acted play structured in three Role-Parts, whereas one Role-Part (RP-1-2) can be skipped by the users based on their decision in a voting (cf. figure 12).

```
<imsld:play identifier="PLAY-1" isvisible="true"> ...
   <imsld:act identifier="Act-1"> <imsld:title>Model a Coin Experiment</imsld:title>
      <imsld:role-part identifier="RP-1-1">...
         <imsld:activity-structure-ref ref="LA-Modelling"/>
      </imsld:role-part>

      <imsld:role-part identifier="RP-1-2">...
          <imsld:learning-activity-ref ref="LA-Further-Modelling"/>
      </imsld:role-part>

      <imsld:role-part identifier="RP-1-3"> ...
          <imsld:learning-activity-ref ref="LA-Presentation"/>
      </imsld:role-part>

      <imsld:complete-act>
         <imsld:when-role-part-completed ref="RP-1-3" />
      </imsld:complete-act>
   </imsld:act>

   <imsld:complete-play>
      <imsld:when-last-act-completed />
   </imsld:complete-play>
</imsld:play>
<imsld:conditions>
   <imsld:if>
        <imsld:is>
            <imsld:property-ref ref="P-Voting"/>
            <imsld:property-value>true</imsld:property-value>
        </imsld:is>
   </imsld:if>

   <imsld:then>
        <imsld:hide>
            <imsld:learning-activity-ref ref="LA-Further-Modelling"/>
        </imsld:hide>
        <imsld:show>
            <imsld:learning-activity-ref ref="LA-Presentation"/>
        </imsld:show>
   </imsld:then>

   <imsld:else>
        <imsld:show>
           <imsld:learning-activity-ref ref="LA-Further-Modelling"/>
        </imsld:show>
        <imsld:hide>
           <imsld:learning-activity-ref ref="LA-Presentation"/>
        </imsld:hide>
   </imsld:else>
  ...
 </imsld:conditions>
</imsld:method>
```

Figure 12: IMS LD Play used in the example script - Note that three Role-Parts are needed since IMS LD does not allow to switch forth and back between acts

Each voting has a title containing the question and at least two options to select from. The options again have a title, refer to an IMS LD property and indicate to what value this property should be set, if this option is the result.

Furthermore the attribute voting-type indicates whether this service is scripted or unscripted. A scripted voting service results in getting configured and placed for the user ready to use, in contrast to an unscripted voting service which just indicates the user has the optional ability to configure his own voting. In Cool Modes the latter will result in the Voting Plugin getting loaded and in case of a scripted voting event a voting node provided by this plugin will get placed on a shared workspace. This creation of a scripted voting can be seen in figure 5 from left (the modelling phase) to right (the voting between the participants).

```
<imsld:environment identifier="LD-Environment1">
    <imsld:title>Modelling Environment</imsld:title>
 <imsld:service identifier="LD-Service1">
    <imsld-ext:voting identifier="LD-Voting1" voting-type="scripted">
       <imsld:title>Is this model correct?</imsld:title>
       <imsld-ext:choice identifier="LD-Choice1" property-ref="P-Voting">
          <imsld:title>Yes</imsld:title>
          <imsld:property-value>1</imsld:property-value> </imsld-ext:choice>
       <imsld-ext:choice identifier="LD-Choice2" property-ref="P-Voting">
          <imsld:title>No incomplete</imsld:title>
          <imsld:property-value>2</imsld:property-value> </imsld-ext:choice>
        <imsld-ext:choice identifier="LD-Choice3" property-ref="P-Voting">
          <imsld:title>No incorrect</imsld:title>
          <imsld:property-value>3</imsld:property-value> </imsld-ext:choice>
    </imsld-ext:voting>
 </imsld:service>
</imsld:environment>
```

**Figure 13:** Environment with a voting service used in the example script

In this case, the environment shown in figure 13 consisting of a voting service is referenced in a learning activity. Thus the learners have the ability to vote if their model needs further work or whether it is already correct and the next modelling phase should be skipped. This works by using IMS LD Level B conditions and showing or hiding the relevant activity based on the value of the property P-Voting. Given a result of a majority for the correctness of the model like in figure 5 a re-iteration of the modelling phase will be skipped and the students directly proceed to the presentation for their classmates.

## 6 Discussion

In this section we want to discuss the presented approach considering the implications of scripted learning environments on learning processes of the learner and problems that usually arise within such systems. Furthermore we want to explain the different variants of using multiple learning environments with or within scripts.

### 6.1 Implications of scripted learning environments

The IMS LD inquiry scripts that we are developing try to tackle a problem that the research community is well aware of [Davis 2003], [Lajoie 2005]. When supporting certain behavior during science learning the outcome of support can be twofold: On the one hand prompts (cf. 5.1) can lead to scaffolding the student within his or her zone of proximal development [Vygotsky 78] leading to a more systematic learning path and triggering thinking processes which might have stayed inactive otherwise. On the other hand however the possibility of disturbing the students own learning path and inhibiting reflective processes through externalized guidance (so called overscripting [Dillenbourg 2002]) seems inevitable. IMS LD scripts allow to adapt the level of coercion achieving the lowest level of support that is needed and the highest level of autonomous learning that is possible. Our approach allows the instructor to apply adaptive support depending on the workflow which then can be mapped flexibly on different learning environments.

To enable the script to decide which level of support is suitable for the learner, *resources and artefacts* that have been resp. produced in earlier phases, should be available to them at later stages to reflect on it and improve their hypotheses in the next cycle. Here it becomes obvious that resources and objects need references through which they can be adressed to be used in multiple phases and cycles of the learning process.

Because the artefacts within the learning process, such as a simulation model created by the student, can change over time, they have to be available both for the LDE and the LSE: the LDE has to initiate the re-appearance of the object in the LSE according to the description of the learning process, while the LSE potentially has to change the content of the object, when the student modifies it in the learning process.

To achieve this flexible use of learning objects in different phases and cycles we chose to represent each re-usable object as a *global personal property* in the LD description. The property is globally defined, because while the initial state of the property can be set in the LD document, the external LSE can manipulate the content during the learning process via the *URI* the property is associated with. A personal property is required here, because every student involved in the learning process might possess an individual version of the artefact, e.g. the simulation model that should be produced in the inquiry process.

### 6.2 Using scripts with different learning environments

The remote control approach allows us to run the same script with different learning environments. While the presented approach supports the usage of two or more different learning environments (e.g. Cool Modes and CoLab) with different students working through the same script, it does not support the automatic launch of these applications. Since the users of the specific collaborative learning environments should be unaware of the fact that the learning session is scripted externally by an LDE, we didn't provide a mechanism that starts a new program on purpose. By this we emphasize the fact that all interaction of the learner is done directly with the learning environment and only indirectly with the learning design engine. Following this philosophy the presented approach allows the remote control approach to use scenarios with different learning environments for the same learner only if the learner started all of them initially. On the other hand no further software installation (i.e apart from the original learning environments) is needed on the client (learning environment) side.

So e.g. if a learning scenario needs a voting facility (see subsection 5.2) and it cannot be provided by the currently active LSEs, no additional voting facility will pop up. But if the same learner is used to such voting scenarios and has a voting facility opened, it is of course possible to control that one too to initiate the script specific voting, given that the voting tool supports the required remote API / Translator. Still there is one flaw with such a setup: all of the collaborating learners will have to use a second application because currently we do not transform the requests for different environments to a request for one single learning environment (merging several tool requests to a singular one) or to more of them depending on the capabilities of the available one (decomposing a request for one tool to several sub-requests for different tools).

A second approach would be to have a special client program that is able to start applications dynamically according to the script. As said before, this would oppose our philosophy of not being intrusive to the user, but it makes the learning processes and the choice of partners for collaboration even more flexible. In [Harrer et al. 2007] we present an approach providing a client that is capable of dynamically launching learning environments. This approach makes use of grid technologies to transfer artefacts between the different learners.

## 7 Conclusion

We have presented a flexible architecture to combine Learning Support Environments with Computer Supported Collaborative Scripting approaches. This is based on our considerations that the pedagogical rationale compiled in scripts has a large potential when transferred to well elaborated (collaborative) learning environments which have been developed in recent years. The idea is to combine the flexibility of learning scripts, which can be adapted to different learning groups and tasks, with the detailed and adaptive support that characterizes task-oriented and domain specific ITS systems. Since the pre-existing systems should not be rewritten substantially we decided to use a loosely coupled approach based on the so-called remote control component that allows to be adjusted for different learning support environments on the one hand and on the other hand different scaffolding agents (e.g. learning design engines, but also human actors) to be applied. We proved our conceptual ideas by presenting the current prototypical implementation of the proposed architecture using the CopperCore engine and the Cool Modes resp. FreeStyler learning environments. The feasibility of our approach for a variety of different learning processes has been shown by the definition and practical test of a complex adaptive script for the support of scientific inquiry learning and a collaborative modelling scenario requiring consensus between the participants. The potential of extending the approach towards the combination of different learning support environments within one learning process and the transferrability of learning resources or products between phases and tools has been elaborated on in the discussion section.

For the specification of the learning processes the Learning Design Standard is used and their are no restrictions with respect to the tools used to create the LD documents. Currently we are working on means for graphical group formation to ease the effort for teachers. For this we plan to use the SessionManager [Kuhn et al. 2005]. Such a graphical specification can help to solve the matter from the user's perspective. Another open question is the definition of an automatic mechanism for group formation in IMS LD scripts. Currently groups are not explicitly represented in the specification, yet there are some workarounds discussed in the literature such as in [Hernandez-Leo et al. 2005], where groups are represented by specific roles. So either substitutes have to be found or new constructs have to be introduced into LD. Following our principle of being as little intrusive as possible, we prefer to extend existing constructs in a way that they conform to the syntax specification of standard IMS LD and that support the semantics needed for automatic group formation.

The second line of research we currently pursue is the transfer of the approach to other collaborative applications with the final goal to create scripted applications which enable the learning designers to specify the interoperability between application rather than programming it. Given this it will be possible to use one learning flow for more than one learning environment at the same time. That means the script (agent, tutor) can be used for other collaborative learning environments, enabling students using different learning environments to collaborate with each other. In the Argunaut project[5] we currently use the Remote API ideas with the third-party collaborative argumentation environment Digalo. There and in the FreeStyler environment (using a visual discussion tool) the remote interventions are enacted by a human moderator to support learners

[5] Project website: `www.argunaut.org`

in their discussions. By providing a mechanism to let a translator publish the set of supported primitives for a LSE, it is possible to allow the dynamic selection of one LSE out of a set of potentially available environments, to perform the execution of the desired functionality, e.g. voting or experimentation. The mechanism has been defined in the Argunaut project and is planned to be put to test in other contexts in future experimentation.

**Acknowledgements**

We thank our student Benedikt Roth for his great job implementing the conceptual framework and architecture designed by us.

We also want to thank the developers of the Reload tool set which enabled us to use the IMS LD standard in our approach.

## References

[Aleven et al. 2004] Aleven, V., McLaren, B.M., Roll, I., and Koedinger, K.R.: Toward tutoring help seeking: Applying cognitive modeling to meta-cognitive skills. In *Proceedings of the Seventh International Conference on Intelligent Tutoring Systems (ITS-2004)*, 227–239, 2004.

[Aleven et al. 2006] Aleven, V., Ashley, K., Lynch, C., and Pinkwart, N. (editors): *Proceedings of the ITS2006 Workshop on Intelligent Tutoring Systems for Ill-Defined Domains*, Jhongli, Taiwan, 2006.

[Amorim et al. 2005] Amorim, R., Lama, M., Sanchez, E., and Vila, X.: A learning design ontology based on the ims specification. In R. Koper, C. Tattersall, and D. Burgos, editors, *Current State on IMS Learning Design. Proceedings of the UNFOLD/Prolearn joint workshop*, 203–225, Valkenburg, 2005.

[Anderson et al. 1995] Anderson, J.R., Corbett, A.T., Koedinger, K.R., and Pelletier, R.: Cognitive tutors: Lessons learned. *Journal of the Learning Sciences*, 4:167–207, 1995.

[Bote-Lorenzo et al. 2004] Bote-Lorenzo, M.L., Vaquero-Gonzalez, L.M., Vega-Gorgojo, G., Dimitriadis, Y., Asensio-Perez, J.I., Gomez-Sanchez, E., and Hernandez-Leo, D.: A tailorable collaborative learning system that combines ogsa grid services and ims-ld scripting. In *Proceedings of the X International Workshop on Groupware, CRIWG 2004*, 305–321. Springer, 2004.

[Dalziel 2006] Dalziel, J.R.: Lessons from LAMS for IMS learning design. In Kinshuk, Rob Koper, Piet Kommers, Paul Kirschner, Demetrios Sampson, and Wim Didderen, editors, *Advanced Learning Technologies ICALT 2006*, 1101–1102, Los Alamitos, CA, 2006. IEEE Computer Society.

[Davis 2003] Davis, E.A.: Prompting middle school science students for productive reflection: Generic and directed prompts. *Journal of the Learning Sciences*, 12(1):91–142, 2003.

[Dawabi and Wessner 2004] Dawabi, P., and Wessner, M.: Modellierung von blended learning szenarien. In *Fachtagung "e-Learning" der Gesellschaft für Informatik*, 115–126, 2004.

[de Jong and van Joolingen 1998] de Jong, T., and van Joolingen, W.R.: Discovery learning with computer simulations of conceptual domains. *Review of Educational Research*, 68:179–201, 1998.

[Dillenbourg 2002] Dillenbourg, P.: Overscripting cscl: The risks of blending collaborative learning with instructional design. In Paul A. Kirschner, editor, *Three worlds of CSCL. Can we support CSCL?*, 61–91. Open Universiteit Nederland, 2002.

[Dillenbourg 2007] Dillenbourg, P.: Split where interaction should happen. In F. Fischer, H. Mandl, J. Haake, and I. Kollar, editors, *Scripting Computer-supported Collaborative Learning - Cognitive, computational, and educational perspectives*, 2007.

[Dillenbourg and Tchounikine 2007] Dillenbourg, P., and Tchounikine, P.: Flexibility in macro-scripts for computer-supported collaborative learning. *Journal of Computer Assisted Learning*, 23:1–13, 2007.

[Forrester 1968] Forrester, J.W.: *Principles of Systems*. Pegasus Communications, Waltham, MA., 1968.

[Harrer et al. 2005] Harrer, A., Hernandez-Leo, D., and Dimitriadis, Y.: Languages for modeling of collaborative learning processes: Formalization, practical uses and tools. Workshop at the Conference on Computer Supported Collaborative Learning, 2005.

[Harrer et al. 2006] Harrer, A., Malzahn, N., and Roth, B.: The remote control approach - how to apply scaffolds to existing collaborative learning environments. In Yannis Dimitriadis and Eduardo Gomez-Sanchez, editors, *Proc. of CRIWG 2006*, volume LNCS 4154 of *Lecture Notes in Computer Science*, 118–131, Berlin, 2006. Springer.

[Harrer et al. 2007] Harrer, A., Lucarz, A., and Malzahn, N.: Dynamic and flexible learning in distributed and collaborative scenarios using grid technologies. In *Proceedings of the XIII International Workshop on Groupware, CRIWG 2007*. Springer, 2007.

[Hernandez-Leo et al. 2005] Hernandez-Leo, D., Asensio-Perez, J.I., Dimitriadis, Y., Bote-Lorenzo, M.L., Jorrin-Abellan, I.M., and Villasclaras-Fernandez, E.D.: Reusing ims-ld formalized best practices in collaborative learning structuring. *Advanced Technology for Learning*, 2(4):223–232, October 2005.

[Jansen 2003] Jansen, M.: Matchmaker tng - a framework to support collaborative java applications. In H.U. Hoppe, F. Verdejo, and J. Kay, editors, *Proceedings of Artificial Intelligence in Education*, 529–530. IOS Press, 2003.

[Kollar et al. 2005] Kollar, I., Fischer, F., and Slotta, J.D.: Internal and external collaboration scripts in web-based science learning at schools. In Timothy Koschmann, Daniel Suthers, and Tak-Wai Chan, editors, *Computer-Supported Collaborative Learning 2005*, 331–340, Mahwah, NJ., 2005. Lawrence Earlbaum.

[Koper and Tattersall 2005] Koper, R., and Tattersall, C. (editors): *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*. Springer, 2005.

[Kuhn et al. 2005] Kuhn, M., Jansen, M., Harrer, A., and Hoppe, H.U.: A lightweight approach for flexible group management in the classroom. In *Proceedings of the International Conference on Computer Supported Collaborative Learning (CSCL2005)*, 353–357, 2005.

[Lajoie 2005] Lajoie, S.P.: Extending the scaffolding metaphor. *Instructional Science*, 33(5-6):541–557, 2005.

[Leitao 2000] Leitao, S.: The potential of argument in knowledge building. *Human Development*, 43:332–360, 2000.

[Mäkitalo et al. 2005] Mäkitalo, K., Weinberger, A., Häkkinen, P., Järvelä, S., and Fischer, F.: Epistemic copperation scripts in online learning environments: Fostering learning by reducing uncertainty in discourse? *Computers in Human Behaviour*, 21(4):603–622, 2005.

[Martel et al. 2006] Martel, C., Vignollet, L., Ferraris, C., David, J.-P. and Lejeune, A.: Modelling collaborative learning activities in e-learning platforms. In Kinshuk, Rob Koper, Piet Kommers, Paul Kirschner, Demetrios Sampson, and Wim Didderen, editors, *Advanced Learning Technologies ICALT 2006*, 707–709, Los Alamitos, CA, 2006. IEEE Computer Society.

[Martens 2004] Martens, A.: Case-based training with intelligent tutoring systems. In *Proceedings of the International Conference on Advanced Learning Technologies*, 191–195, 2004.

[Martens and Uhrmacher 2004] Martens, A., and Uhrmacher, A.M.: A formal tutoring process model for intelligent tutoring systems. In *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI 2004*, 124–128, 2004.

[McLaren et al. 2005] McLaren, B.M., Bollen, L., Walker, E., Harrer, A., and Sewall, J.: Cognitive tutoring of collaboration: Developmental and empirical steps towards realization. In *Proceedings of the Conference on Computer Supported Collaborative Learning Conference (CSCL-05)*, 418–422, 2005.

[O'Donnell et al. 1992] O'Donnell, A.M., and Dansereau, D.F.: Scripted cooperation in student dyads: A method for analyzing and enhancing academic learning and performance. In R. Hertz-Lazarowitz and N. Miller, editors, *Interaction in cooperative groups: The theoretical anatomy of group learning*, pages 120–141. Cambridge University Press, 1992.

[Pinkwart 2005] Pinkwart, N.: *Collaborative Modelling in Graph Based Environments.* PhD thesis, University of Duisburg-Essen, 2005.

[Quintana et al. 2002] Quintana, C., Krajcik, J., and Soloway, E.: A case study to distill structural scaffolding guidelines for scaffolded software. Technical report, CHI, 2002.

[Reload] RELOAD: Re-using e-learning object authoring & delivery. http://www.reload.ac.uk/, 2007. Accessed online on Dec 14th 2007.

[Ritter and Koedinger 1996] Ritter, S. and Koedinger, K.R.: An architecture for plug-in tutor agents. *International Journal of Artifical Intelligence in Education*, 7(3/4):315–347, 1996.

[Suthers et al. 1995] Suthers, D., Weiner, A., Connelly, J., and Paolucci, M.: Belvedere: Engaging students in critical discussion of science and public policy issues. In J. Greer, editor, *Proceedings of AI-ED 1995*, 266–273, 1995.

[van Joolingen et al. 2004] van Joolingen, W.R., Lazonder, A.W., de Jong, T., Savelsbergh, E.R., and Manlove, S.: Co-lab: Research and development of an online learning environment for collaborative scientific discovery learning. *Computers in Human Behavior*, (21):671—688, 2004.

[Vega-Gorgojo et al. 2006] Vega-Gorgojo, G., Bote-Lorenzo, M.L., Gomez-Sanchez, E., Asensio-Perez, J.I., Dimitriadis, Y., and Jorrin-Abellan, I.M.: Ontoolcole: an ontology for the semantic search of cscl services. In *Proceedings of the 12th International Workshop on Groupware. Springer-Verlag, LNCS 4154 (CRIWG 2006)*, 310–325, Medina del Campo, Spain, September 2006.

[Vogten et al. 2005] Vogten, H., Koper, R., Martens, H., and Tattersall, C.: An architecture for learning design engines. In R. Koper and C. Tattersall, editors, *Learning Design*, pages 75–90. Springer, 2005.

[Vygotsky 78] Vygotsky, L.: *Mind in Society.* Harvard University Press, Cambridge, MA, 1978.

[Weinberger 2003] Weinberger, A.: *Scripts for Computer-Supported Collaborative Learning Effects of social and epistemic cooperation scripts on collaborative knowledge construction.* PhD thesis, University of Munich, 2003.

[Weinberger et al. 2005a] Weinberger, A., Fischer, F., Häkkinnen, P., Dillenbourg, P., and Harrer, A.: Computer-supported scripting of interaction in collaborative learning environments. Workshop at the Conference on Computer Supported Collaborative Learning, 2005.

[Weinberger et al. 2005b] Weinberger, A., Stegmann, K., and Fischer, F.: Computer-supported collaborative learning in higher education: Scripts for argumentative knowledge construction in distributed groups. In Timothy Koschmann, Daniel Suthers, and Tak-Wai Chan, editors, *Computer-Supported Collaborative Learning 2005*, 717–726, Mahwah, NJ., 2005. Lawrence Earlbaum.

[WISE] WISE - The web-based inquiry science environment: `http://wise.berkeley.edu`, 2007. Accessed online on Dec 14th 2007.