2010 International Conference on Intelligent Networking and Collaborative Systems

Closing the Circle: IMS LD Extensions for Advanced Adaptive Collaboration Support

Florian König, Alexandros Paramythis
Institute for Information Processing and Microprocessor Technology (FIM)
Johannes Kepler University
Altenbergerstraße 69, A-4040 Linz, Austria
{koenig, alpar}@fim.uni-linz.ac.at

Abstract—This paper, the third in a series, completes the presentation of a proposed set of modifications and extensions to the IMS Learning Design specification with the goal of enabling better support for adaptivity in collaborative learning settings. The extensions presented here target advanced adaptation features that build upon previous work and include: adapting control flows; controlling adaptations on a meta-level; human involvement in adaptation decisions; "transactional" action processing; loops and branches for controlling action execution; declaration of re-usable action sequences and complex expressions; and mechanisms for exception handling.

Keywords—collaborative learning; adaptive support; learning design; IMS LD; extension; adaptation actions; control flow; adaptation control; provisional adaptation decision; exception handling

I. INTRODUCTION

Learning is regarded by many theorists as a social activity that can benefit from a collaborative setting [1]. In collaborative learning situations, certain interactions (such as discussion, mutual questioning, joint problem solving, conflict, collective sense-making and shared agreement) can trigger learning mechanisms and result in improved learning outcomes [2]. There is, however, no guarantee that beneficial interactions occur, even less so in distance learning, where lack of or limited 'real-world' contact amongst learners can negatively influence social- and group learning- patterns [3]. Especially in collaborative e-learning, support is needed to increase the probability that the desired interactions occur, because teams may not have worked together before, are usually formed for a comparatively short time, and individual learning goals are predominant. One way to support learners is by *scaffolding* their interaction in order to get group work going, mitigate disorientation and reduce cognitive load [4]. *Collaboration scripts* [5] are an instance of such a scaffolding strategy. They provide a detailed specification of the collaboration contract in a scenario, and aim to set up systematic differences among learners in order to trigger contentious interactions, or to make rich interactions for exchanging complementary knowledge necessary [2]. For computer-supported collaborative learning (CSCL), collaboration scripts are modelled and enacted via external, computational representations, termed *CSCL scripts* [6]. These contain instructions specifying how members of a group should interact and collaborate to attain a task [7].

Static collaboration scripts, however, represent idealized scenarios and there is a danger of impeding fruitful collaboration by an overly coercive script (*over-scripting*) [5]. One strategy of adjusting scripts to learners and groups is to reduce its scaffolds over time in a process called *fading* [8], once learners have a good understanding of how to collaborate effectively. To adjust scripts at run-time in general and target the individual needs of learners and groups, *adaptive collaboration scripting* has been proposed [9],[10] and found to be an effective method [11].

The most widely known formalization for CSCL scripts is the IMS Learning Design (IMS LD) specification [12], a learning process modelling language intended to formally describe designs of teaching-learning processes for a wide range of pedagogical approaches [13]. Still, IMS LD has been criticized both for insufficient expressiveness in aspects of the collaborative learning process [6], and for the absence of constructs that are vital in supporting adaptivity [14]: the language provides limited support to model group-based, synchronous collaborative learning activities and collaboration contexts [6]; the specification of services for providing, amongst others, collaboration facilities has been found to be rather inflexible [15]; IMS LD is missing an event model to monitor state changes and trigger adaptations, and it has only very few functions to modify a collaboration process at run-time [14]; artefacts can not be explicitly specified; IMS LD exposes no run-time model for querying state information and effecting changes; the activity sequencing model has been found to be constraining [16] and difficult to understand [17]; and, there is no mechanism to handle exception states that may arise at run-time.

Based on extensions of IMS LD that have already been developed (section II) or are proposed in the literature (section III), we present modifications and additions to the specification, aiming to address the aforementioned shortcomings and improve support for adaptivity in CSCL scripts. Specifically, the amendments deal with actions for adapting the control flow (section IV.A), control of adaptations on a meta-level (section IV.B), human involvement in adaptation decisions (section V), transactional action processing, loops and branches for controlling action execution (section VI), declaration of re-usable action sequences and complex expressions (section VII) and mechanisms for exception handling (section VIII). Section IX summarizes the goals attainable through the proposed extensions, and provides an outlook on issues to be worked on in future iterations.

978-0-7695-4278-2/10 $26.00 © 2010 IEEE
DOI 10.1109/INCOS.2010.83

II. EXTENSIONS TO IMS LD

In order to address shortcomings of IMS LD precluding its effective use in supporting collaborative and adaptive learning (see section I), certain extensions to both its information model and its run-time behaviour have already been developed and described in detail [18],[19]. As these extensions form the basis for the work presented here, their most important aspects will be summarised in the remainder of this section. A simplified model of the run-time objects and their inter-relations can be seen in Fig. 1. Elements added to or changed from IMS LD are shown in gray. The scripting capabilities have been extended as follows: Groups can be explicitly modelled (*group*), either statically or via specifying constraints for run-time creation of dynamic groups. Group members can share group-specific properties and an environment that acts as a common group workspace. A choice of different policies gives control over the grouping of participants for populating groups at run-time (*grouping*). For roles, properties and shared workspaces can be specified as well. Casting participants into roles is now possible at run-time via policies similar to those for grouping (*casting*). Grouping and casting operations can be sequenced by means of two new activities: *grouping-activity* and *casting-activity*.

To avoid relying on the properties for storing individual and group work results, artefacts can be modelled explicitly, and their flow between different learning activities is expressed by referencing them as input-, output- or transient-artefacts. Coupled with permissions, this defines how they progress in the script process and who can contribute. For collaboration context provision, the *service* specification has been enhanced to support a wide range of services, constraint-based auto-selection and fine-grained permissions.

In the new model, the *method* can have multiple *story* containers (corresponds to the IMS LD element *play*), which can each have multiple *scene* elements (correspond to combination of *act* and *role-part*). Scenes tie actors (members of *role* objects) to activities (*learning-activity*, *grouping-activity*, *casting-activity*) and are instantiated at various granularities: one *scene-instance* for all actors, one per group or one per actor. In combination with environments, which may provide communication and collaboration services, collaboration contexts for groups and classes can be created at the process level. Scenes are connected to each other via *transition* objects, allowing for arbitrary sequencing of activities and making loops possible. Workflow semantics are used for splitting and synchronizing the flow of action.

To implement advanced adaptation features in a script, knowledge of its run-time state is required. The proposed extensions to IMS LD feature a run-time model with access to all elements of the script. This model also contains pure run-time data, such as information on participants. Data from the run-time model can be used in expressions, which made some operators redundant (e.g., *users-in-role*) and new ones necessary (e.g., *runtime-value*). More flexible and expressive definitions of adaptation rules are made possible through an event handling mechanism with event-condition-action (ECA) semantics (*event-handler*). Each object in Fig. 1 shown with an asterisk can be monitored by an event handler. Events are also generated by services; this allows reacting to behaviour of participants in external tools. Event handlers can trigger a multitude of new adaptation actions, which can manage the life-cycle of objects (creation, destruction), their attributes and relation to other objects, and can intervene in the environment, to enable and disable resources and services, or to directly control external services.

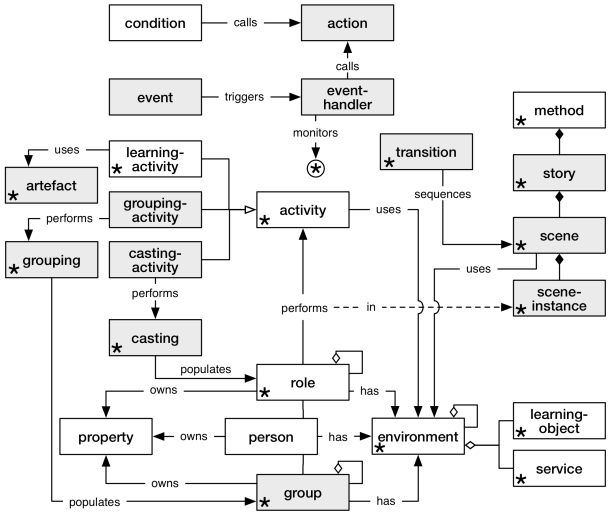

Figure 1. Object model (partial). New objects in gray.

III. RELATED WORK

IMS LD provides some support for adapting a scenario, of which Towle and Halm [20] give an overview. Four areas where a scenario can be tailored to individual learners are identified: available resources (*environment*), sequencing of activities (*method*), different rights and obligations (*roles*), and actual tasks to perform (*activities*). The description of how to implement these adaptations, however, already shows limitations of IMS LD like the difficulty to express complex adaptations, hard-coded and unstructured adaptation rules, no possibility for re-using rules and lack of a run-time model.

A number of extensions have been proposed in recent years to address shortcomings of IMS LD discussed in section I. Berlanga and García [21] present a framework which covers tests on learning style and knowledge, student modelling, and strategies for tailoring a scenario to individual students expressed in adaptation rules. Zarraonandia et al. [22] describe an approach for introducing small variations into learning design scripts at design- and run- time. The proposed adaptation actions modify activities, environments, the structuring of these elements, resources, properties, and completion criteria. Advanced adaptations like the introduction of new roles or acts, complex conditions and major structural changes are, however, not discussed.

To improve support for collaborative scenarios, Miao and Hoppe [23] propose an extension for modelling groups, relying on operations for creating/deleting groups, member management and run-time state queries. A mechanism for defining complicated expressions and actions is introduced, and

the need for a loop control structure for action declarations is put forth. The authors also advocate a semi-automated approach for involving humans in adaptation decisions.

Miao et al. [6] present a comprehensive collection of ideas on how to improve support for adaptive collaboration scripting. The proposed extensions include explicit modelling of artefacts, a more flexible sequencing model with workflow semantics, means to access more run-time state information and contexts for social interaction. With regard to adaptation, the authors propose to extend IMS LD with operations concerning activities, artefacts, roles, groups, persons, transitions, environments and relations between them.

Other suggestions for adaptation actions in the literature include varying the group size, recommending or assigning (changes in) roles for participants, modifying the activity structure (e.g., adding / removing / reordering tasks), determining the availability of elements (activities, services, artefacts) [14], invoking system facilities and manipulating the system state (e.g., initiating communication sessions). Paramythis and Cristea [24] also present some more requirements for adaptation languages in the area of collaboration support, none of which are supported by IMS LD: workflow- or process- based reasoning, temporal operators, policies for grouping, and support for "provisional" adaptation decisions.

IV. Adaptation Actions

The original IMS LD specification provides only a limited number of actions: showing/hiding objects, changing the value of properties and giving feedback. Comprehensive support for adaptive interventions, however, requires a wide range of additional actions in order to effect meaningful changes on the execution of a learning design. The necessary actions can be broadly categorized into the following classes:

- *object adaptations*: creating, modifying and destroying objects (group, scene, artefact, …)
- *relation adaptations*: modifying attributes of objects or processes in relation to other objects or processes (membership, ownership, permissions, visibility, …)
- *control flow adaptations*: starting, stopping, modifying (e.g., new branches) the script process
- *environment adaptations*: managing services and performing human-like actions in the environment
- *adaptation control*: managing the adaptations themselves (conditions, event handlers, actions, …)

These actions cover the life-cycle of objects, relations between them, control of active processes, adaptations of external services and basic building blocks for meta-adaptivity. Two of these classes will now be described in more detail.

A. Control Flow Adaptations

Adapting the control flow deals with modifying the sequencing of activities in scenes and stories, instantiation and completion of sequencing elements, and controlling their run-time state. In our model, transitions sequence the scenes in a story by connecting them and modelling simple sequences, branches or joins of the control flow, and loops. Possible adaptations to transitions are: setting a condition, to make the transition conditional; modifying an existing condition; and removing the condition, to convert to an unconditional transition. Depending on a scene's configuration of how its outgoing control flow is split among the transitions connecting it to subsequent scenes (single outgoing, multiple outgoing, all outgoing), there are constraints on the number of required conditional and unconditional transitions [18]. If, for instance, the control flow should be split to all subsequent scenes, only unconditional transitions can be used. Conversely, when adaptively changing the flow split mode for the outgoing control flow of a scene, the types of outgoing transitions may have to be adjusted as well. These actions need to be executed in combination (see section VI), so as not to violate constraints. Another adaptation action is required to change how many preceding scenes (one, multiple, or all) need to transfer the control flow to start a scene.

Collaboration contexts are formed by creating one instance of a scene for all participants (class context), for each group of a set of groups (group context), or per participating person (personal context). This instantiation configuration may be changed by an adaptation action, but only before a scene has started. In the case of group contexts, a set of groups or a reference to a grouping is needed as parameter.

Completion criteria can be changed at run-time for activities, scenes, scene instances, stories and the method. After the action has been performed, the respective element can become immediately completed if the criterion is fulfilled.

Finally, there are adaptation actions to directly control the state of the control flow: *start*, *pause*, *resume*, *complete*, and *cancel*. Depending on the object on which they are applied, they have different effects. Pausing, resuming, completing and cancelling a story also performs that operation on the (running) scenes it contains and propagates to scene instances. Starting a story works at any time and causes it to run in parallel to and independent from other stories. Starting a scene creates the scene instances, instantiates the environment, transfers existing artefacts from preceding activities according to the specification and starts activities whose control mainly rests with the system and not the participants (grouping-activity, casting-activity). Scene instances can be started individually in the context of a running scene but require a list of actors. Pausing a scene (instance) disables the associated event handlers, suspends the timers [19], locks the artefacts currently used in it, and makes the state change visible to participants. Resuming a scene (instance) reverses this process. Completing a scene stops all its instances, marks it as complete, notifies the participants that the scene has been completed and passes the control flow to its outgoing transition(s), which (depending on the sequencing configuration) lets participants continue the story. Cancelling a *scene* works in the same manner but marks it as cancelled.

B. Adaptation Control Actions

These actions are very different from the others, because they do not adapt those parts of a learning design script that directly influence its participants but rather modify the specification and run-time enactment of adaptive interventions themselves. They are necessary for effecting meta-adaptivity (i.e., adaptation of a system's own adaptive behaviour), also referred to as a second-level adaptation cycle [25]. There are multiple ways of influencing adaptivity in a script.

Conditions, as defined in IMS LD, are a set of *if-then-else* rules, and cannot be enabled or disabled at run-time, except by using a custom local property (e.g., *rule5-enabled*) to explicitly control this aspect. By introducing container elements for such rules, referring to individual (sets of) conditions becomes possible. Run-time access to these containers, coupled with actions to enable and disable them, allow for basic modifications of a script's primary adaptation logic.

In order to support reacting to more state changes, structure the rule base and improve semantics, an event handling mechanism has been introduced [18]. At run-time, *event-handler* objects can be created and added to (and removed from) objects that support them (*scene*, *role*, *service*, etc.) Enabling and disabling event handlers works like it does for rules. For triggering them, *event* objects can be created and "injected into" objects supporting event handlers or directly into a specific event handler. Depending on its filter expression, a handler will react by executing its actions.

For implementing adaptation control actions, a meta-model of the script, its rules, event handlers and actions is needed. It also has to be noted, that these actions only provide the infrastructure for realizing interventions. The complex problem of assessing the resulting effects and the reasoning processes required for deciding when and how to modify the adaptation strategy are not part of this work.

V. Provisional Adaptations

Script authors may want to express that some adaptations are not automatically enacted, but are left for a human (e.g., a teacher) to decide. These decisions can be binary (enact specific action or not) or multi-valued (enact one of many choices). This approach is similar to what Dieterich et al. [26] term *user-controlled self-adaptation*: the system takes the initiative, creates and presents an adaptation proposal to a user who decides, and the system takes over again to execute the user's choice. The difference here is, that participants do not necessarily decide about adaptations that influence themselves but also those that influence others.

In order to make adaptations provisional, the respective actions can be 'wrapped' in a container element, that specifies details like who is asked and what happens if this person did not respond. Any list of actions can be made provisional and used like a normal action in, for instance, conditions or event handlers. It is possible to specify a single list of actions, which would allow choosing whether to enact this sequence or not, or a set of choices of different lists of actions, from which one can be chosen. Each action list needs to have a (human-readable) description that explains its intention and effect. The whole provisional action may have a description. In combination with context information by the system, these descriptions are used to generate the information presented to the decision makers, who are identified via their roles. The following modes of decision are currently supported:

- one member of the role(s) can decide alone;
- all members of the role(s) have to agree: either everybody, or all members who are currently online;
- a simple majority of role members suffices: again, either all or just the members who are online.

One problem with provisional adaptations is that they may never be reacted upon, because notifying the decision makers was not possible, they did not respond, or they could not agree. To remedy such problems, a timeout can be defined, after which one of the following fallback options can be put into effect: the adaptation is cancelled altogether; a default choice (specified in the choice of actions) or the default option of the single action (execute/do not execute) is enacted; the decision is escalated to a different role and a new timeout starts with different fallback options.

Like other elements of the model, provisional adaptations can be adapted (see section IV.B). Before they are activated, the choice of actions can be changed. They can also have event handlers, through which state changes can be monitored and acted upon. While running, the list of roles tasked with the decisions, the decision mechanisms and the fallback action definition can be changed. Other possible actions include cancelling a provisional adaptation, enacting the default, triggering escalation or escalating to a different role.

VI. Action Control Structures

In a lot of cases, action execution is constrained in a variety of ways, mostly to ensure referential integrity between script elements. There are, however, operations required during adaptation, where the individual steps would violate such constraints but the end result is valid again. An example of this is mentioned in section IV.A, where a change to the sequencing configuration of a scene requires specific transitions to originate from it, but modifying those transitions can only happen in other, subsequent actions. Modifying the transitions beforehand would also not work, because this would again violate the constraints of the sequencing configuration. In order to make this adaptation possible, actions can be grouped in an action block. Each action block works like a transaction: it is regarded as an atomic operation, which either completes in its entirety or fails because its accumulated effects violate constraints, which are only checked after executing the block. Action blocks contain nested actions and can be used in all places where actions can be used.

Advanced control over the execution of sets of adaptation actions requires two structures known from traditional programming languages: branches and loops. For modelling branches, the structure of the original IMS LD conditions [12] can be reused, because it already specifies the required *if-then-else* scheme, where a Boolean expression is evaluated and the actions in the respective branch are called. With loops, a list of actions can be repeated as long as an expression evaluates to *true*. The *while-do* scheme is used.

VII. Composite Actions and Named Expressions

Compared to the original IMS LD specification, many adaptation actions and expression operators have been added in the extensions proposed so far, in order to increase the expressiveness and modelling capabilities of learning design scripts [18],[19]. The majority of those, however, work at a rather low level, dealing with just one object and performing one operation (not counting the effects from those on other objects). For instance, adding a branch with a scene to an ex-

isting control flow requires: creating the scene, connecting it with transitions to the scene where the branch occurs and to the scene where the branched control flow should join the main one again, and setting the transition conditions so that the desired actors get to the new scene. Even this simplified description already shows that atomic operations need to be combined to result in meaningful, higher-level effects. For this reason, we propose to extend IMS LD with means to create composite actions and named expressions as combinations of other (composite) actions and (named) expressions. This makes it possible to define often-required operations once and re-use them across a script. Authoring tools or run-time engines could also provide libraries of such operations.

Composite actions require a unique name with which they are invoked, a list of actions to perform and (optionally) parameters. When invoked, the actions are performed in sequence. Parameters allow abstracting the declaration from concrete situations in which the action is used. Each parameter has a name, a type and a flag which specifies whether it is mandatory. For optional parameters a default value can be given, which is used when no value is provided by the caller. Parameters can be accessed inside the declaration like any run-time model data item [18] via its name in the special, local scope *$params* (e.g., *$params.paramName*).

Named expressions are declared in a similar manner. The only difference is that instead of the list of actions, the respective expression needs to be declared. Parameters can be used as well. Upon invocation, the named expression is evaluated and its result is returned.

VIII. Exception Handling

With the shift to more adaptive and dynamic learning design scripts, a lot of data is processed at run-time and actions may constantly modify the scenario. These operations can lead to situations and states that were not anticipated while authoring the scenario and could result in a lock-up or termination of the script. Similar to event handlers [18], we propose to extend IMS LD with a mechanism to handle those exception states. There is, however, an important difference between events and exceptions: Events can be handled when the script author deems it important or helpful for detecting state changes. Exceptions, on the other hand, cannot be left unhandled; there must be at least a default handler. If, for instance, the instantiation of a service (e.g., chat tool) fails, this could be handled by trying a different service or changing the scenario so that the service is not needed any more. If no such custom exception handling was defined, a default handler must take appropriate actions such as stopping the script and notifying an administrator. Exception handlers can be defined for individual objects; this provides them with maximum context and intercepts problems immediately where they appear. Default handlers can also be specified for the whole script, and, as a last resort, have to be provided by the run-time engine. Exceptions can happen in a lot of cases, but in general the following categories can be distinguished:

Participant exceptions

- *actors missing*: roles having to perform actions (activity, review, …) or taking part in operations requiring their members, are not populated (any more)
- *groups missing*: required groupings and groups are empty or have been disbanded
- *insufficient participation*: participants did not act in the required way (e.g., voting for a team leader)

Action exceptions

- *constraint violated*: operation not allowed by constraints (referential integrity, range, multiplicity, etc.)
- *operation rejected*: operation (casting, grouping, action, etc.) rejected by reviewer
- *operation not possible*: operation cannot be performed on object (e.g., cancel a completed scene)

Sequencing exceptions:

- *incomplete sequencing*: sequencing model is incomplete (e.g., missing transitions) while story is running

Data exceptions

- *data type mismatch*: data type do not match in assignment or literal value does not fit requirements
- *run-time model*: object in run-time model is not accessible or writable
- *expression invalid*: run-time model access expression has wrong syntax

Constraint solving exceptions

- *no-result*: constraints are not satisfiable (e.g., when grouping, casting, auto-selecting services)

External exceptions

- *service-unreachable*: external service (e.g., collaboration or grouping service) not reachable
- *service-failed*: operation in external service failed (e.g., service deployment)

These are just the main types of exceptions that can occur and not the individual exceptions, which are too numerous to list and describe here in detail.

Defining exception handlers works in a similar manner to specifying event handlers. Unlike the later, however, exception handlers cannot filter through expressions when they are to be triggered; instead, they are triggered every time one of the specified exceptions occurs. Each exception has a run-time property, which contains a reference to the object in which it was caused, and a human-readable description of the cause, which can for example be used when notifying somebody. An exception handler also defines whether the exceptions should be fully handled by itself, or subsequently propagated to the default handlers on the script or run-time engine level. This makes it convenient, for instance, to handle exceptions where they occur and use a catch-all handler at script-level to send notifications to relevant people.

IX. Summary and Outlook

This paper constitutes the last instalment in a series that puts forward a comprehensive set of extensions to IMS LD intended to provide extensive and grounded support for adaptation in collaborative learning designs. The proposed extensions are currently under evaluation using criteria such as the ones described in [27]. In addition, a representative selection of existing, well-known collaboration scripts is being used to assess whether all required features are supported and can be expressed.

In parallel, we have commenced work on a prototypical implementation of the extended specification into the Sakai

e-learning platform [28]. This will then be employed in real-world student-based evaluations, where we will seek to establish the impact of the newly enabled types of adaptive support on the collaborative learning process. Following that, we intend to turn our attention towards tool-oriented support for authoring CSCL scripts in the extended specification, with a focus on facilitating the utilization of common strategies, patterns, and templates even by educators with little prior experience in the area.

ACKNOWLEDGMENTS

The work reported in this paper has been supported by the "Adaptive Support for Collaborative E-Learning" (ASCOLLA) project, financed by the Austrian Science Fund (FWF; project number P20260-N15).

REFERENCES

[1] J. Roschelle and S.D. Teasley, "The construction of shared knowledge in collaborative problem solving," *Computer-Supported Collaborative Learning*, C. O'Malley, Ed., Berlin: Springer, 1995, pp. 69–97.

[2] P. Dillenbourg, "What do you mean by collaborative learning?" *Collaborative-learning: Cognitive and Computational Approaches*, P. Dillenbourg, Ed., Oxford: Elsevier, 1999, pp. 1–19.

[3] A. Paramythis and J.R. Mühlbacher, "Towards New Approaches in Adaptive Support for Collaborative e-Learning," *Proceedings of the 11th IASTED International Conference*, Crete, Greece: 2008, pp. 95–100.

[4] J. Zumbach, J. Schönemann, and P. Reimann, "Analyzing and supporting collaboration in cooperative computer-mediated communication," *Proceedings of the 2005 conference on Computer support for collaborative learning: learning 2005: the next 10 years!*, Taipei, Taiwan: International Society of the Learning Sciences, 2005, pp. 758–767.

[5] P. Dillenbourg, "Over-scripting CSCL: The risks of blending collaborative learning with instructional design," 2002.

[6] Y. Miao, K. Hoeksema, H.U. Hoppe, and A. Harrer, "CSCL Scripts: Modelling Features and Potential Use," *Proceedings of the 2005 Conference on Computer Support for Collaborative Learning – Learning 2005: The next 10 Years!*, Taipei, Taiwan: International Society of the Learning Sciences, 2005, pp. 423–432.

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

[8] R.D. Pea, "The social and technological dimensions of scaffolding and related theoretical concepts for learning, education, and human activity," *The Journal of the Learning Sciences*, vol. 13, 2004, pp. 423–451.

[9] N. Rummel, H. Spada, and S. Hauser, "Learning to collaborate while being scripted or by observing a model," *International Journal of Computer-Supported Collaborative Learning*, vol. 4, Mar. 2009, pp. 69–92.

[10] N. Rummel, A. Weinberger, C. Wecker, F. Fischer, A. Meier, E. Voyiatzaki, G. Kahrimanis, H. Spada, N. Avouris, E. Walker, K.R. Koedinger, C.P. Rosé, R. Kumar, G. Gweon, Y. Wang, and M. Joshi, "New challenges in CSCL: Towards adaptive script support," *Proceedings of the 8th International Conference of the Learning Sciences – Volume 3*, Utrecht, The Netherlands: International Society of the Learning Sciences, 2008, pp. 338–345.

[11] S. Demetriadis and A. Karakostas, "Adaptive Collaboration Scripting: A Conceptual Framework and a Design Case Study," *Complex, Intelligent and Software Intensive Systems, International Conference*, Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 487–492.

[12] IMS Global Learning Consortium, Inc., "Learning Design Specification (Version 1.0 Final Specification)," 2003.

[13] R. Koper and B. Olivier, "Representing the Learning Design of Units of Learning," *Educational Technology & Society*, vol. 7, 2004, pp. 97–111.

[14] A. Paramythis, "Adaptive Support for Collaborative Learning with IMS Learning Design: Are We There Yet?," *Proceedings of the Adaptive Collaboration Support Workshop, held in conjunction with the 5th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'08)*, Hannover, Germany: 2008, pp. 17–29.

[15] M. Caeiro, L. Anido, and M. Llamas, "A Critical Analysis of IMS Learning Design," *Proceedings of CSCL 2003*, Bergen, Norway: 2003, pp. 363–367.

[16] J. Torres and J.M. Dodero, "Analysis of Educational Metadata Supporting Complex Learning Processes," *Metadata and Semantic Research*, 2009, pp. 71–82.

[17] K. Hagen, D. Hibbert, and P. Kinshuk, "Developing a Learning Management System Based on the IMS Learning Design Specification," *IEEE International Conference on Advanced Learning Technologies (ICALT 2006)*, Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 420–424.

[18] F. König and A. Paramythis, "Towards Improved Support for Adaptive Collaboration Scripting in IMS LD," *Sustaining TEL: From Innovation to Learning and Practice – Proceedings of the 5th European Conference on Technology Enhanced Learning Sustaining (EC-TEL 2010)*, M. Wolpers, P.A. Kirschner, M. Scheffel, S. Lindstädt, and V. Dimitrova, Eds., Barcelona, Spain: Springer Verlag, 2010, pp. 197–212. (in press)

[19] F. König and A. Paramythis, "Collaboration Contexts, Services, Events and Actions: Four Steps Closer to Adaptive Collaboration Support in IMS LD," *Proceedings of the International Conference on Intelligent Networking and Collaborative Systems (INCoS 2010)*, Thessaloniki, Greece: 2010. (in press)

[20] B. Towle, M. Halm, R. Koper, and C. Tattersall, "Designing Adaptive Learning Environments with Learning Design," *Learning Design. A Handbook on Modelling and Delivering Networked Education and Training*, Springer-Verlag, 2005, pp. 215–226.

[21] A.J. Berlanga and F.J. Garcia, "A Proposal to Define Adaptive Learning Designs," *Proceedings of Workshop on Applications of Semantic Web Technologies for Educational Adaptive Hypermedia (SW-EL 2004)*, Eindhoven, The Netherlands: Technische Universiteit Eindhoven, 2004, pp. 354–358.

[22] T. Zarraonandia, J.M. Dodero, and C. Fernández, "Crosscutting Runtime Adaptations of LD Execution," *Educational Technology & Society*, vol. 9, 2006, pp. 123–137.

[23] Y. Miao and U. Hoppe, "Adapting Process-Oriented Learning Design to Group Characteristics," *Proceeding of the 2005 conference on Artificial Intelligence in Education: Supporting Learning through Intelligent and Socially Informed Technology*, IOS Press, 2005, pp. 475–482.

[24] A. Paramythis and A. Cristea, "Towards Adaptation Languages for Adaptive Collaborative Learning Support," *Proceedings of the First International Workshop on Individual and Group Adaptation in Collaborative Learning Environments (WS12) held in conjunction with the 3rd European Conference on Technology Enhanced Learning (EC-TEL 2008). CEUR Workshop Proceedings, ISSN 1613-0073, online CEUR-WS.org/Vol-384/*, Maastricht, The Netherlands: 2008.

[25] P. Totterdell and P. Rautenbach, "Adaptation as a problem of design," *Adaptive user interfaces*, Academic Press, 1990, pp. 61–84.

[26] H. Dieterich, U. Malinowski, T. Kühme, and M. Schneider-Hufschmidt, "State of the Art in Adaptive User Interfaces," *Adaptive User Interfaces*, M. Schneider-Hufschmidt, T. Kühme, and U. Malinowski, Eds., Elsevier Science Publishers B. V, 1993, pp. 13–48.

[27] M. Caeiro-Rodríguez, M. Llamas-Nistal, and L. Anido-Rifón, "Towards a Benchmark for the Evaluation of LD Expressiveness and Suitability," *Journal of Interactive Media in Education*, 2005.

[28] Sakai Foundation, *Sakai Project*, http://www.sakaiproject.org.