Towards Improved Support for Adaptive Collaboration Scripting in IMS LD

Florian König and Alexandros Paramythis

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

Abstract. The IMS Learning Design specification is acknowledged as the most promising option available presently for the implementation of collaboration scripts in e-learning. Nevertheless, it has been criticized for a number of shortcomings, and, specifically for its lack of support for constructs that would enable comprehensive adaptive support to be effected over the collaborative learning process. In this paper we propose concrete extensions to the specification, which build upon prior work and address a wide range of problems and omissions. The most important modifications introduced include: explicit support for groups, and run-time member assignment; addition of a run-time model; introduction of concrete artefacts; introduction of an event-handling model; and, a modified sequencing and script organization model.

Keywords: collaborative learning, adaptive support, learning design, IMS LD, extension, sequencing, grouping, artefact

1 Introduction

It is widely acknowledged that a large part of the success of the learning process lies with the opportunities of learners to interact with others: group work, exchanging ideas, and helping each other (thereby learning oneself) are standard "classroom" practices [1]. The establishment of online learning as a viable alternative to "traditional" approaches has resulted in the progressive lifting of geographical, temporal and other barriers to learning; but, at the same time, it has introduced obstacles in approaching learning as a social activity, due to the limited contact between learners that absence of physical collocation inevitably incurs [2].

The field of Computer-Supported Collaborative Learning (CSCL), which first appeared as a specialized direction in the area of Computer-Supported Collaborative Work (CSCW), is at the dichotomy between CSCW and e-learning, and seeks to support the various forms of collaborative learning in online settings [3]. A core focus of CSCL work is on ways in which collaborative learning supported by technology can enhance peer interaction and work in groups, and ways in which collaboration and technology facilitate sharing of knowledge and expertise among community

members. A promising axis of work within CSCL is the scaffolding of productive interactions between learners by specifying in detail the 'collaboration' contract in a scenario [1], in so-called CSCL scripts [4]. The later are external, computational representations of collaboration scripts, which are sets of instructions specifying how the members of a group should interact and collaborate to solve a particular problem [5].

The work reported in this paper falls within a line of research that seeks to further the principles underlying the use of CSCL scripts to guide the collaborative learning process, by introducing the dimension of adaptivity [6]. In this context, adaptivity concerns the provision of automated support within learning processes that involve communication between multiple learners (and, therefore, social interaction), and, potentially, collaboration towards common objectives. Furthermore, the support provided should be tailored to the specific requirements of both the participating individuals, and the social units in which they are involved (e.g., groups undertaking course projects). The goal of this line of work can then be stated as enabling the incorporation within CSCL scripts of constructs that would allow for expressing the types of adaptive support that a system can automatically offer to learners in the context of collaborative learning activities.

Unfortunately, the main starting point towards the above stated goal, namely a general modelling language for formalising collaboration scripts, is still missing [4]. At present, the most promising effort in that direction is the IMS Learning Design (IMS LD) specification [7], which evolved out of the Educational Modelling Language developed by the Open University of the Netherlands [8]. IMS LD is a learning process modelling language, fashioned on the theatrical play metaphor (e.g., with plays, actors, roles, etc.), and intended to formally describe any design of teaching-learning processes for a wide range of pedagogical approaches [8, 9].

IMS LD has been criticized both for its lack of support for sufficiently expressing aspects of the collaborative learning process [4], and for the absence of constructs that are vital in supporting adaptivity in scripts [10]. This paper builds upon that criticism, as well as on proposals for extensions of the specification that have appeared in the literature (discussed in the next section), to propose a comprehensive set of modifications and additions to IMS LD, aiming to address many of its general shortcomings and lay the foundations for improved support for adaptivity in CSCL scripts.

The rest of the paper is structured as follows. Section 2 presents an account of shortcomings of IMS LD and proposals for extensions that have appeared in the literature. Section 3 outlines the modifications and extensions proposed, relating them to two exemplary adaptive collaborative learning scenarios. Finally, section 4 summarizes the goals that can be achieved through the proposed amendments, and provides an outlook on issues that we want to address in future iterations.

2 Related Work

The IMS Learning Design modelling language has its strengths in specifying personalized and asynchronous cooperative learning. In the area of (adaptive) collaboration scripting, however, a number of shortcomings have been identified [4]: Modelling

multiple groups independent from their role(s) is possible in IMS LD only with limitations, and manipulating groups at run-time is not supported. Miao and Hoppe [11] introduced an extension for modelling groups, which relies on newly defined operations for creating and deleting groups, member management and run-time state queries. However, they do not provide a high-level specification mechanism for run-time grouping, which would make it easy for both authors and adaptation engines to express and understand the semantics of group (membership) management.

In IMS LD there are also no means to specify how the members of a group interact within an activity, other than through a so-called "conference service" [12]. Hernández-Leo et al. [13] proposed the "groupservice" for specifying group collaboration spaces with communication facilities, floor control, support for different interaction paradigms and awareness functionality. This approach, however, suffers from the same limitations as the original IMS LD specification: the actual collaboration happens outside the script's specification and control. Miao et al. [4] proposed running multiple instances of activities, if required by the respective social plane (one per role/group/person) to allow, for example, groups to work in parallel on the same problem. Their approach, however, is still not geared towards maximum expressive flexibility. Even with these multiple instance activities, IMS LD provides insufficient support for (collective) artefacts [4] and no straightforward mechanism to model the flow of artefacts between activities [14]. Therefore, proposed already to some extent by Caeiro et al. [15], Miao et al. [4] also defined a possible model for artefacts, where the data flow is derived from the specification of creation-/consume- activities.

IMS LD's activity sequencing capabilities have been described as "quite simplistic" [16] and fine-grained splitting or synchronization of the control flow is not readily expressible [4]. Caeiro et al. [15] addressed some shortcomings by introducing a different model for the role-part in IMS LD. It would link elements according to the guiding question of who (roles) does what (act) in which way (operational role) where (environment) and towards which objective (outcome). The operational role would be independent from a person's roles in order to make varying responsibilities possible. They also proposed having conditional transitions between acts to allow more complex sequencing. Miao et al. [4] reiterated the need for advanced transition and routing constructs guided by approaches in the area of workflow management.

The features discussed until now form the basis for collaboration scripting. With respect to explicit support for adaptivity, IMS LD is missing a model of the run-time state of the script, and an event model, and has only a limited number of functions to modify aspects of the collaboration process while the script is running [10]. Paramythis and Christea [17] 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 clustering, invoking system facilities / manipulating system state and support for "provisional" adaptation decisions. In our approach, we address a number of these requirements and lay the foundations for supporting more in the future. The following section describes our proposed extensions to IMS LD. We aimed to cover the aforementioned collaboration support requirements and the requirements with regard to adaptivity.

3 Proposed Extensions to IMS LD

Our approach combines ideas from many of the extensions proposed in related work, puts them into a coherent information model and introduces additional features that are intended to better support adaptivity. It encompasses the modelling of groups, run-time group and role membership assignment (section 3.1), group properties and property collections, a run-time model, an adapted and extended set of operators (section 3.2), an extensible modelling of services (section 3.3), artefacts, event handling, and new actions for effecting the scenario at run-time (section 3.4) as well as a flexible sequencing model (section 3.5). The relevant changes are depicted in diagrams of the information model, to be found in Fig. 1, 2 and 3, and resembling the diagrams in the original IMS LD specification [7]: child elements are either in a sequential (orthogonal connectors) or a choice (angular connectors) grouping; they are optional (question mark symbol) or required once (no symbol), zero or more times (star symbol), or one or more times (plus symbol). In the process of making the diagrams as space-efficient as possible, the order of the elements could not always be kept the same as in the textual description. Due to the complexity of both the original and the extended model, it is advisable to follow the diagrams in parallel to the explanations.

To explain our extensions in the context of real examples, we will use two well-known collaborative learning scenarios, JigSaw [18] and Thinking Aloud Pair Problem Solving (TAPPS) [19], that are often implemented as CSCL scripts. Both examples require features that cannot be modelled with the original IMS LD specification. Table 1 summarizes the unique requirements of each scenario, the limitations of IMS LD, and in which section the extension supporting the requirement is proposed.

Table 1. Limitations of IMS LD support for unique requirements of scenaria.

Scenario	Requirement	IMS LD limitations	Extension
JigSaw	run-time grouping by characteristics	no concept of groups	section 3.1
	run-time casting of group leader	roles assigned before start	section 3.1
	(group) collaboration contexts	not in process model no group workspaces	section 3.1 and 3.5
	explicit artefacts, flow of artefacts	only properties	section 3.4
	flexible event handling (e.g., detecting when artefact delivered)	unstructured set of conditions with no semantics	section 3.4
TAPPS	service selection by criteria	only 3 types of "conference" service specified	section 3.3
	role rotation between participants	roles fixed during run	section 3.4
	multiple executions of activity (loop)	rigid sequencing model	section 3.5

In JigSaw, learners are grouped into mixed (ability, knowledge, …) groups of 5-6 members and a group leader is appointed. The material is split into 5-6 topics and each group member gets assigned to one. Students read their material individually and then come together in "expert groups" (one per topic), where they discuss their parts and how they are going to present them to their JigSaw groups. The students then re-

unite in the JigSaw groups, present their topic, and group peers can ask questions. In the end, each group collaboratively solves an assignment that involves all topics. The instructor collects the assignments and provides feedback. In TAPPS, students are paired and given a list of problems. One gets assigned the role of "problem solver", the other becomes the "listener". The problem solver tries to solve the problem while thinking aloud. The listener follows, identifies errors and asks questions. After solving a problem, they exchange roles and move on to the next problem.

3.1 Groups, Roles and Run-Time Member Assignment

In IMS LD, grouping participants can only be simulated by assigning them to predefined, static roles. Roles are expected to be populated before the unit of learning starts and role membership does not change thereafter. In our approach, groups can be directly modelled, and assignment of participants to both groups and roles is possible at run-time. There are statically defined groups and dynamic groups, which are created at run-time according to certain constraints (e.g., preferred size). Static groups are defined by *group* elements, which reside in *groups* in the *components* part of the script (see Fig. 1). Each group has a name (*title*) and can optionally have *information*, for example regarding its purpose. A *group-environment* (referencing *environment* elements) can be specified to give a group its own "group space", for example with communication facilities and shared material. In the JigSaw example, there would exist a group for each topic (the "expert group") with the material provided in the respective *group-environment*. Sub-groups can be defined via nested *group* elements.

The specification for creating dynamic groups and assigning participants to (static and dynamic) groups is located in *grouping* elements (see Fig. 1). A grouping consists of a set of groups and a mapping of participants to them. Groupings may be requested from an external provider (*provided-grouping*) or created at run-time (*runtime-grouping*). The first option allows re-using groupings, which exist for example in the LMS running the script. For a *runtime-grouping* the *partition* specifies, into which groups participants should be grouped. This can be a *static-group-set* containing references to existing *group* definitions or a *dynamic-group-set* with a set of constraints according to which groups are created automatically at run-time: The *number-of-groups* can be constrained to an *exact* number, a range (*from*, *to*) and/or to be *divisible-by* a specific value. The *group-size* element can be used to express equivalent constraints, so for the pairing in the TAPPS example an *exact* value of 2 can be set. In addition, one can specify a set of *proportions* to create, for example, groups with the size ratios of 1:2:4. Dynamic groups are automatically named and sequentially numbered (*group-names*) by specifying a *prefix* and/or *suffix* string around a *numbering* of *latin*, *roman* or *alphabetical* style (e.g., "Group III", "2. Gruppe", "Grupo k").

The *grouping-method* defines how the participants are assigned to groups. For a *manual* assignment, a *role* has to be referenced, which will be tasked to perform it manually. With *self-selected* assignment participants can themselves choose in which group they would like to be: a *direct* assignment is immediate; in a *prioritized* assignment participants can attach numeric priorities to each group to mark their (least) favoured choice and will then be assigned by balancing their preferences.

A totally *automatic* assignment can be *random*. Another possibility is *by-grouping-service* where a (possibly external) grouping service can be used to run a certain *grouping-strategy* with a set of *parameter* elements. This makes it possible to employ arbitrarily complex grouping algorithms and provide them with the required feature vector(s) of determinants. In the JigSaw example, where mixed groups are required, an algorithm could create them when provided with parameters detailing the relevant attributes (knowledge, interests, etc.) The *automatic* method *by-existing-grouping* can re-use the mapping of participants in *existing-groups* –represented by a *grouping* or a *static-group-set*– to create new groups. It is possible to *distribute* the members (approximately) equally over the new groups or *concentrate* them (i.e., keep them together). This is useful for example in JigSaw, where the JigSaw group members should be equally distributed across the groups representing the topics. With the method *role-selectable*, grouping methods can be specified, from which, at run-time, a member of the referenced *role* can choose one to perform the grouping.

For a *runtime-grouping* a *review* can be defined, which will present members of a referenced *role* with a certain *environment* to allow reviewing the result of the *grouping*. Like for single groups, a *group-environment* can be defined. Each group will be provided with its own group space according to this definition, which can also be set for a *provided-grouping*. Should a (static) group already have a group space, then the two will be merged with the one defined in *group* taking precedence.

The original definition of *roles* in IMS LD was kept, but a way has been added to specify how the assignment (i.e., "casting") of participants to them is to be performed at run-time. Similar to *grouping* there is a *casting* element, which also resides in the *method* part of the script (see Fig. 1). Under *roles-to-cast* multiple roles can be referenced, which will all get the same *role-environment*, if specified. The *casting-method* is similar to the *grouping-method*. With *by-vote* all eligible participants can vote (a) candidate(s) into a role either by a *direct* vote or *prioritized* by their preference to the candidates. The automatic *by-casting-service* method works like the *by-grouping-service* as described above, with one addition: for each role, multiple *role-requirement* elements can be used to specify criteria that prospective role members should fulfil. In JigSaw, this could contain information such as: the leader role should have an authority level above a certain threshold. Like for a grouping, a *review* can be defined for a casting as well. Depending on the scenario, not all roles can be cast at run-time, however. Any roles that need to be populated so that the learning-design can start must be referenced in the *initial-roles* element under *learning-design*.

3.2 Properties and Run-Time Model

Storing and accessing data inside a learning design using IMS LD is accomplished via properties. We have extended properties to account for groups and the need for collections (lists, sets). Additionally, we have defined a run-time model and means to use it in expressions, giving adaptation rules access to the run-time state of a scenario.
Groups can have a *locgroup-property*, which is local to the current run, as well as a *globgroup-property*, which is kept across runs. The owning group(s) need(s) to be specified in *owner* (see Fig. 1).

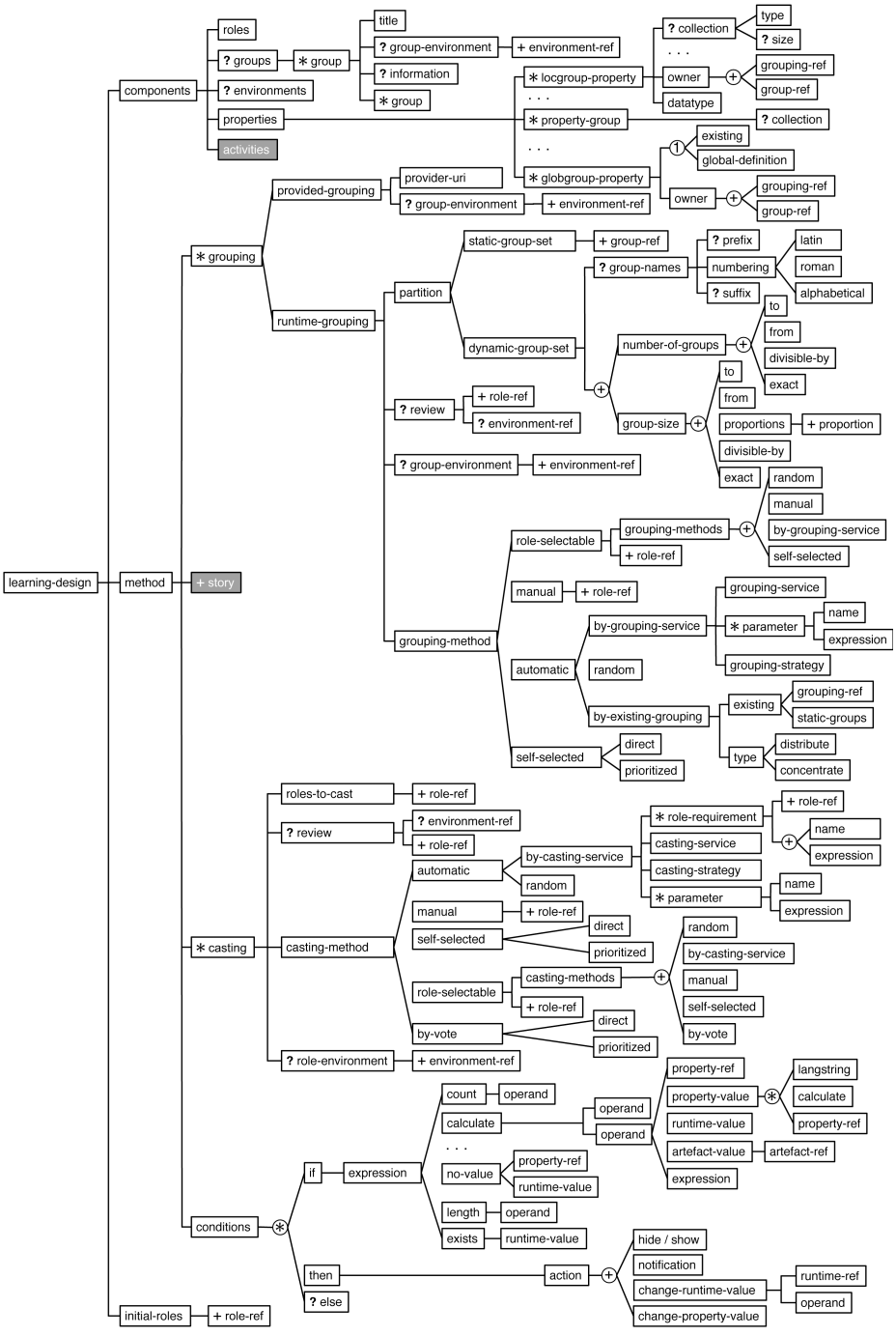

Fig. 1. Extensions to IMS LD (gray elements shown in following figures).

Static groups exist in every run and can have both local and global properties. Dynamically created groups can own only local properties, as the set of groups may change in every run. Groups of a *provided-grouping* may own global properties, which they keep as long as the external group exists.

Some extensions not related to collaboration have been added to the property model as well. They apply to every type of property, even though in Fig. 1 they are only shown in the context of the *locgroup-property*. Property and property group specifications now feature an optional *collection* element, where the type (ordered *list* or unordered *set*) can be specified. The *size* defines the maximum number of elements to be either *dynamic* or fixed to the result of an *expression*. The resulting property (group) is a collection of its defined data type. In order to constrain the content of a property of type *file*, the *restriction* now supports *mimeType* as a *restriction-type*. It takes a comma-separated list of MIME types (e.g. "text/html, text/xml, text/plain").

IMS LD allows only very limited access to run-time information via operators like *is-member-of-role*, *complete* or *time-unit-of-learning-started*. Most run-time modelling requires properties, but these custom-made models are not standardized or readily re-usable across scripts. The addition of a well-defined run-time model makes it possible to access a large part of the state information of a running script. The run-time model is structured like the static learning design and every statically defined element and attribute can be accessed. References are automatically resolved and the respective elements appear to contain the referenced ones. In addition, a range of run-time-only elements and attributes is added to the model (e.g., *members* in a *learner* role). One important (exclusively) run-time element, reachable for example via *members*, is *person*, which represents a participant. There is also a *system* element, which contains information about the run-time environment (e.g., *current-datetime*).

To access information inside the run-time model, the element *runtime-value* was added under *expression* in the *conditions* part as a new type of *operand* and in the *no-value* test operator. It expects a *run-time model access expression* following this syntax: *type[selector].property*. The *type* corresponds to the names of top-level-elements like *environment*, *learning-activity* or *group* and specifies which element(s) to access. In cases where there is no common type (e.g., *learner*, *staff*) the following "virtual" types can be used: *role*, *activity*, *property*, *condition*. If there exist more elements of the requested type in the run-time model, the *selector* can filter them with a logical or positional expression. Possible logical operators are =, *!*=, <, <=, >, >=, *&* (and), | (or) and *!* (not). Parentheses can be used to group terms, and the precedence rules of the ANSI C programming language apply. Properties of elements to be filtered are accessed as *@property*. Literal values need to be enclosed in quotes (e.g., *role[@title = "leader"].members*). Positional expressions with integer results can be used as zero-based index in collections and do not need to be quoted (e.g., *group[@id = "grp1"].members[2]*). It is also possible to request collections, perform operations on them (see the end of this section), store them in collection properties (see above) or use them as parameters for adaptation actions (e.g., creating a group of participants which were retrieved from the run-time model). Selectors can contain nested run-time model access expressions. If the *property* in the expression is itself a structured element, it can be further filtered. Nested elements are accessed via the dot operator.

For writing to the run-time model, everywhere in the information model where *change-property-value* is permitted, the new element *change-runtime-value* can be used. It requires a *runtime-ref* containing a run-time model access expression pointing to the property that should be set. The value comes from an *operand* element.

Because the run-time model is dynamic, a new operator *exists* was introduced to check whether an object referenced by a run-time model access expression is present. When access to the run-time model is not guarded by an accompanying check with *exists*, accessing non-existing elements can result in exceptions. Therefore, script authors need mechanisms to detect exceptions and react accordingly. The specification of these exception handling mechanisms will be addressed in future work.

The following obsolete operators of the original IMS LD specification could be removed, because their results are now contained in the run-time model: *is-member-of-role*, *users-in-role*, *time-unit-of-learning-started*, *datetime-activity-started*, *current-datetime* and *complete*. The standard arithmetic *modulo* operator has been introduced, as well as a range of set (*contains*, *union*, *intersection*, *complement*) and temporal (*precedes*, *meets*, *overlaps*, *equals*, *includes*) operators, which have been omitted from the diagram for reasons of space efficiency. To support collections, a *count* operator for getting the number of elements has been added. The number of characters in a text string can be determined with the new *length* operator.

3.3 Services

The definition of services in the IMS LD specification has been criticized as being rather inflexible [15]. Our approach builds upon the *Generic Service Integration* approach of Valentin et al. [20] and provides three ways to specify a service: direct reference (URI), type selection (by a descriptor like *service:chat*) or constraint-based (requesting for example a direct, synchronous, speech-oriented communication service for the thinking aloud sessions in the TAPPS example). As the service specification is, very important in general, albeit not of direct relevance to the immediate support of collaboration in a learning design, we will not go deeper into this topic here.

3.4 Activities, Artefacts, Event-Handling and Actions

Grouping participants and casting them into roles conceptually fits neither in a *learning-activity* nor in a *support-activity*. Therefore, we have created two new activity types. A way to specify artefacts as one of the basic building blocks of collaborative work has been added as well. In order to support fine-grained means to react to run-time events, a mechanism to define event handling and perform actions is proposed.

Like for learning activities, there needs to be an element for describing acts of grouping participants or casting them into roles, that can be referenced when defining the sequencing of a learning design. We have extended the IMS LD specification by a *grouping-activity* and a *casting-activity* (see Fig. 2). Both reference a *grouping* or a *casting* respectively, can use a certain *environment* and have an *activity-description*. Their mode can be set to either “start from scratch” (*create-grouping* / *cast-roles*) or to *add-members* or *remove-members*, where existing groupings / castings need to be

expanded or shrunk. These activities allow for creating groups and populating groups and roles. Disbanding groups and removing roles is part of the adaptation actions that will be defined in future iterations of this extension to IMS LD.

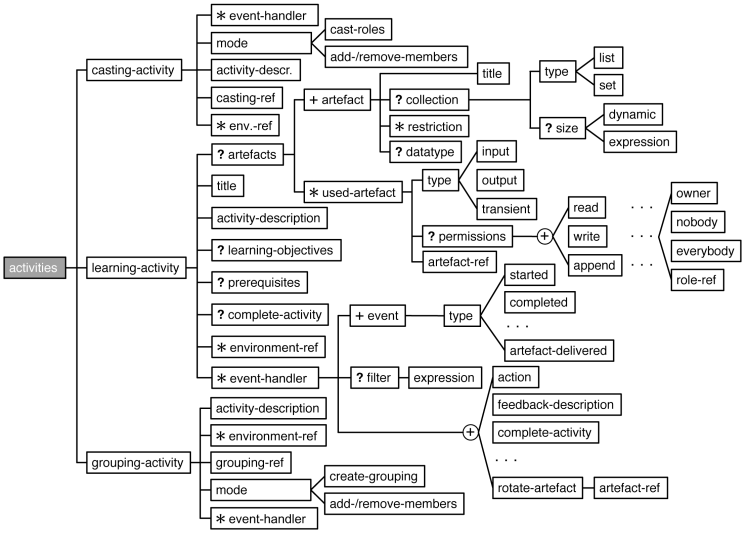

Fig. 2. Extensions to the *activities* element of IMS LD.

In the original IMS LD specification, the results of work performed by participants are stored in and retrieved from properties. Access to them is possible via any referenced XHTML document through the use of special *global-elements* tags. The script itself, however, did not contain any explicit information about which result had to be created in which activity, and where else this result was used. In our approach, *artefacts* can be specifically defined in a *learning-activity*. An *artefact* has a *title* and the author can specify whether the artefact is a *collection*, or whether any other *restriction* applies (see definition of properties). If an artefact should be visible in a certain activity, it can be referenced as a *used-artefact*. An *output* artefact can be set during the activity, one of type *input* is only shown, and *input-output* artefacts can be both accessed and changed. Artefacts are owned by the entity that created them (participant or group). The default *permissions* (*read*, *write*, *append*) belong to the *owner*. However, it is also possible to give those permissions to either *nobody*, *everybody* in the current activity or members of a certain *role*. In the JigSaw example, students would create an output artefact as a solution to the assignment in one activity. In the next activity, this artefact could be used as an input artefact, and the instructor role could be given read access. The feedback artefact created by the instructor would be handled in a similar way. Finally, like properties, an artefact may be used in an *expression*. It can be either referenced directly as an *operand* via an *artefact-ref* or its value be read when referencing it inside the *artefact-value* element (see Fig. 2).

With the *on-completion* element in IMS LD one can specify that certain actions should be performed (e.g., *change-property-value*) when something (e.g., an activity)

is completed. Two restrictions exist in this approach: it is not possible to perform an action when something starts (or some other event occurs); and, there are actions (e.g., *show* and *hide* for controlling the visibility of elements) that cannot be triggered. The first restriction was tackled by replacing *on-completion* with *event-handler* elements, each able to handle a certain event (e.g., *started*, *completed*, *artefact-delivered*, …). The second restriction was resolved by collecting common actions in an *action* element in the *then* (and *else*) part of a *condition* (see Fig. 1). In an *event-handler*, any of those actions may be triggered. Depending on the element in which the handler is defined, additional actions (e.g., *complete-activity*) can be defined. The list of possible events is also specific to the surrounding element. Every *event-handler* can also have a *filter* with an *expression* to specify the exact conditions when it triggers actions. In the *expression*, the run-time event can be accessed via the special construct *$event*. Each event has a *timestamp* property and, depending on its type, it may have additional properties (e.g., *activity*: to access the activity that has completed). A complete list of events and their properties cannot be provided here due to lack of space.

Comprehensive support for adaptive interventions requires a large number of actions in order to change most parts of the run-time model and effect changes on the execution of the learning design. A number of actions has already been added to address shortcomings of IMS LD. Event handlers of a *learning-activity* can for example trigger a *rotate-artefact* action to effect a circular exchange of the referenced artefact among the entities (single participants or whole groups) currently performing the activity. The *rotate-roles* action rotates roles among participants of a certain *casting* (which must have assigned more than one role) or the members of a set of groups. In TAPPS, for each pair there would be a casting for the "problem solver" and "listener" roles. With the *rotate-roles* action, the roles can be switched between the members of each pair. In addition to these two examples, many other actions need to be defined, for example for creating and removing roles, groups, environments and activities, adding to and removing members from roles and groups, managing properties and altering the sequencing of activities, to name just a few.

3.5 Sequencing

For complex scenarios and even more so for adaptive collaboration scripts, which require flexible execution flows, the sequencing semantics of the original IMS LD specification have been found to be constraining [21] and difficult to understand [22]. Due to the lack of state-of-the art concurrency control features, one would often have to resort to custom-made mechanisms employing properties and conditions. In our approach, we have aimed at simplifying the constructs needed for sequencing, while at the same time supporting groups, allowing arbitrary transitions between activities and employing flexible concurrency controls from the area of workflow management.

The original metaphor of a theatrical play has been replaced by the following concept (see also Fig. 3): in a *story* there are *scene* elements, sequenced by directional transitions connecting them. Each scene has *actors* defined by their *role*, who perform a *task*: an *activity* or alternatively a nested *story*. In addition to the *environment* in the *activity*, one can be specified for each scene as well, possibly as an override.

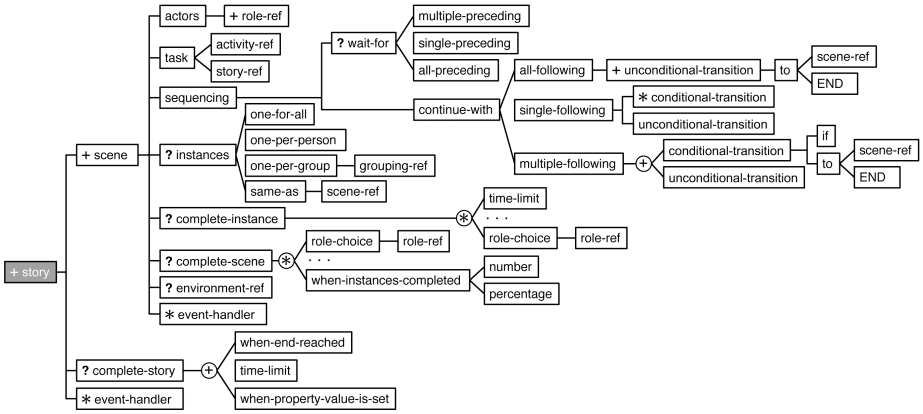

Fig. 3. Elements for sequencing activities.

The multiplicity of a scene is a central concept of our approach. When a scene is run in a single instance (*one-for-all*), all actors share the same activity and, through it, environment and artefacts. Alternatively, a scene can be split into multiple instances: *one-per-person* or *one-per-group* with a respective reference to a *grouping*. For JigSaw, the reading and expert group scenes would be split by topic grouping, whereas the topic presentations and assignment solving scenes would be split to give each JigSaw group a separate instance. TAPPS pairs also act in individual instances. By specifying *same-as*, the multiplicity is inherited from the referenced *scene*. The multiplicity also defines how artefacts are aggregated or distributed: artefacts created in the activity of a multi-instance scene are collected and made available when they are used in the activity of a subsequent single-instance scene. Artefacts originating from a single-instance scene and used in a multi-instance scene, are copied to each instance. Other transfers are only possible when the multiplicity of both scenes matches.

Our model of sequencing has been influenced by common workflow control-flow patterns [23]. These represent generally acknowledged solutions to frequently encountered sequencing requirements in flow-oriented process models. Torres and Dodero [21] have found IMS LD to be lacking with regard to expressing certain workflow patterns. By modelling the patterns' basic building blocks as well as supporting some of them directly, improved support could be attained. Individual *scene* elements are connected by transitions to form a general graph. This provides maximum flexibility and makes it possible to model loops (required for example in the TAPPS scenario, see Fig. 4), which are not supported by IMS LD. As discussed by Gutierrez-Santos et al. [24], we also needed to amend the original semantics, so that activities can be completed multiple times when they are instantiated more than once in a loop.

Depending on the flow split mode, set via *continue-with* in *sequencing*, elements of *unconditional-transition* or *conditional-transition* are needed. The latter require an *if* element with a logical *expression* that evaluates to *true* to activate the transition. Both types can lead either to a referenced *scene* or (through the special value *END*) to the end of the current story. Using the flow split mode of *all-following* splits the flow to all outgoing transitions (AND-split). With *single-following*, the flow follows the first

conditional-transition that has been set active by its condition, or alternatively the mandatory *unconditional-transition* (XOR-split). In *multiple-following* mode, the execution flow continues along all active transitions, resulting in multiple outgoing flows (OR-join). Conversely, with the element *wait-for* the mode of synchronizing incoming execution flows can be specified: It is optional by default and *all-preceding* is assumed, which synchronizes all incoming flows (AND-join). With *single-preceding*, the flow continues after arriving from a single incoming transition (XOR-join). The *multiple-preceding* mode has the following semantics: execution continues after one or more incoming flows have been received and there is no incoming transition left, over which another flow could arrive in the future (OR-join). This last mode, albeit useful and used for implementing a number of workflow patterns, is not straightforward to implement because it requires non-local semantics [23].

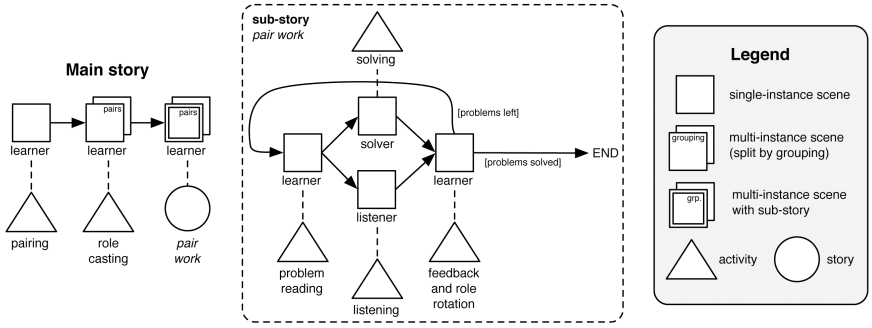

Fig. 4. Simple representation of TAPPS [19] with new sequencing semantics.

By default, an instance of a scene (be it the only one or one of multiple) completes when its activity or nested sub-story completes. In the *complete-instance* element, however, multiple completion criteria can be defined. The options have been carried over from the IMS LD element *act*, and a new element has been added: *role-choice* gives members of the specified role a possibility for manual intervention. These options are also valid choices for the *complete-scene* predicate element. Additionally, with *when-instances-completed* one can specify how many or what percentage of instances are required for scene completion. This might be useful to express the following scenario: learners solve a problem and as soon as 80% have solved it, the whole class moves on to look at common solutions.

4 Summary and Outlook

In this paper, we have presented extensions to the IMS Learning Design specification that aim to better express collaboration scripts and allow for more comprehensive adaptation to individual learners and groups. The new elements of the information model support explicit modelling of (static) groups, their properties and group environments. Assignment of participants to static, as well as to dynamically created

groups can be controlled via a number of policies; the same is true for the assignment to roles. Activity types for these grouping and role casting operations have been introduced. The extensions support a run-time model, giving access to all elements of the script and pure run-time data, such as information on participants, while it is running. Access to the run-time model is possible in expressions, making some operators obsolete. New set and temporal operators have been added. The service specification has been made more flexible to account for the plethora of possible services in today's learning management systems. Artefacts can be modelled explicitly, may be assigned a data type, and have restrictions associated with them, like properties. In the context of a learning activity, artefacts may be used as input, output or transient elements, optionally with access permissions (read, write, append) for different participants. An extensible event handling specification with event-condition-action (ECA) semantics has been introduced, and new actions (e.g., rotate artefacts, rotate roles) for adapting the scenario at run-time have been defined. Finally, the original sequencing model has been replaced with a more flexible one, which supports running multiple instances of activities according to the social plane (class, group, individual), offers (conditional) transitions for arbitrary sequencing of activities and uses workflow semantics for synchronizing and splitting the flow of action.

For future iterations of this information model, we plan to fine-tune its current features and provide new ones to enhance its expressiveness towards improved adaptivity. First, the run-time model needs to be fully and thoroughly specified, providing clear semantics for every element, its (data) type and multiplicity and introducing a temporal dimension to allow access to historic state information. Exception handling mechanisms need to be introduced to protect against script failure when non-existing elements of the run-time model are accessed. The run-time model needs to be made accessible from the client side (assuming the script is "executed" on the server side), in a way that conceptually extends (and ideally replaces) the *global-elements* of IMS LD. The main idea is to provide an implementation-independent interface specification similar to the SCORM run-time API [25], to allow for bi-directional data transfer between the learning design engine and the client (application). Another requirement is access from within the learning design script to external models like, for example, the personal learner models of participants in the learning management system. Making these models accessible via the respective element in the run-time model (e.g., person) would render them immediately usable by the mechanisms described so far.

For effecting adaptations, more actions are needed, especially for re-sequencing scenes, invoking system facilities (e.g., notifying participants) and manipulating the system state (e.g., controlling services). As suggested by Miao and Hoppe [11], a way to create expressions and action declarations needs to be modelled, in order to allow re-using sets of actions and complicated expressions. To support provisional adaptation decisions, a mechanism must exist to specify that certain participants (e.g., the instructor) should be consulted about whether to apply a specific adaptation or not. One of the benefits of such a model is that the instructor does not need to be present at all times, but can still monitor and control the scenario. Finally, the event model needs to be made more fine-grained to capture all possible events that occur during the run of a learning design and may be relevant for adaptation rules. This includes a basic

event model for the different service types, so that participants' actions in them (e.g., chat entered, message posted, wiki page edited) can be reacted upon.

Once we have arrived at a mature specification that includes the additional elements described above, we intend to integrate a prototypical implementation of it into the Sakai e-learning platform [26]. This will then be employed in real-world student-based evaluations, where we will seek to establish the impact of the types of adaptive support that the new specification enables on the collaborative learning process.

Acknowledgements. 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. Dillenbourg, P.: What do you mean by collaborative learning? In: Dillenbourg, P. (ed.) Collaborative-learning: Cognitive and Computational Approaches. pp. 1–19. Elsevier, Oxford (1999).
2. Paramythis, A., Mühlbacher, J.R.: Towards New Approaches in Adaptive Support for Collaborative e-Learning. Proceedings of the 11th IASTED International Conference. pp. 95–100, Crete, Greece (2008).
3. Lipponen, L.: Exploring foundations for computer-supported collaborative learning. Proceedings of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community. pp. 72–81, International Society of the Learning Sciences, Boulder, Colorado (2002).
4. Miao, Y., Hoeksema, K., Hoppe, H.U., Harrer, A.: CSCL Scripts: Modelling Features and Potential Use. Proceedings of the 2005 Conference on Computer Support for Collaborative Learning – Learning 2005: The next 10 Years! pp. 423–432, International Society of the Learning Sciences, Taipei, Taiwan (2005).
5. O'Donnell, A.M., Dansereau, D.F.: Scripted Cooperation in Student Dyada: A Method for Analyzing and Enhancing Academic Learning and Performance. In: Hertz-Lazarowitz, R. and Miller, N. (eds.) Interaction in Cooperative Groups: The theoretical Anatomy of Group Learning. pp. 120–141,Cambridge University Press, London (1992).
6. Jameson, A.: Adaptive interfaces and agents. The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications. pp. 305–330, L. Erlbaum Associates Inc. (2003).
7. IMS Global Learning Consortium, Inc.: Learning Design Specification (Version 1.0 Final Specification), http://www.imsglobal.org/learningdesign/, (2003).
8. Koper, R.: Modeling units of study from a pedagogical perspective: the pedagogical meta-model behind EML, http://hdl.handle.net/1820/36, (2001).
9. Koper, R., Olivier, B.: Representing the Learning Design of Units of Learning. Educational Technology & Society. 7, 97–111 (2004).
10. Paramythis, A.: 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). pp. 17–29, Hannover, Germany (2008).
11. Miao, Y., Hoppe, U.: 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. pp. 475–482, IOS Press (2005).
12. Santos, O.C., Boticario, J.G., Barrera, C.: Authoring A Collaborative Task Extending the IMS-LD to be Performed in a Standard-Based Adaptive Learning Management System Called aLFanet. Proceedings of the Workshop on Adaptive Hypermedia and Collaborative Web-based Systems (AHCW'04) held in conjunction with the International Conference on Web Engineering (ICWE 2004). Munich, Germany (2004).
13. Hernández-Leo, D., Perez, J., Dimitriadis, Y.: IMS Learning Design Support for the Formalization of Collaborative Learning Patterns. Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT 2004). pp. 350–354 (2004).
14. Miao, Y., Burgos, D., Griffiths, D., Koper, R.: Representation of Coordination Mechanisms in IMS Learning Design to Support Group-based Learning. Handbook of Research on Learning Design and Learning Objects: Issues, Applications and Technologies. pp. 330–351, IDEA Group (2008).
15. Caeiro, M., Anido, L., Llamas, M.: A Critical Analysis of IMS Learning Design. Proceedings of CSCL 2003. pp. 363–367, Bergen, Norway (2003).
16. Dalziel, J.: From Re-usable E-learning Content to Re-usable Learning Designs: Lessons from LAMS, http://www.lamsinternational.com/CD/html/resources.html, (2005).
17. Paramythis, A., Cristea, A.: 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).
18. Aronson, E., Blaney, N., Stephin, C., Sikes, J., Snapp, M.: The Jigsaw Classroom. Sage Publishing Company, Beverly Hills, CA (1978).
19. Lochhead, J., Whimbey, A.: Teaching analytical reasoning through thinking aloud pair problem solving. New Directions for Teaching and Learning. 1987, 73–92 (1987).
20. de la Fuente Valentin, L., Miao, Y., Pardo, A., Delgado Kloos, C.: A Supporting Architecture for Generic Service Integration in IMS Learning Design. Times of Convergence. Technologies Across Learning Contexts. pp. 467–473 (2008).
21. Torres, J., Dodero, J.M.: Analysis of Educational Metadata Supporting Complex Learning Processes. Metadata and Semantic Research. pp. 71–82 (2009).
22. Hagen, K., Hibbert, D., Kinshuk, P.: Developing a Learning Management System Based on the IMS Learning Design Specification. IEEE International Conference on Advanced Learning Technologies (ICALT 2006). pp. 420–424, IEEE Computer Society, Los Alamitos, CA, USA (2006).
23. Russell, N., Arthur, van der Aalst, W., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center (2006).
24. Gutierrez-Santos, S., Pardo, A., Kloos, C.D.: Authoring Courses with Rich Adaptive Sequencing for IMS Learning Design. Journal of Universal Computer Science. 14, 2819–2839 (2008).
25. Advanced Distributed Learning Initiative: Sharable Content Object Reference Model (SCORM) 2004 4th Edition Version 1.1 – Run-Time Environment, http://www.adlnet.gov/Technologies/scorm/, (2009).
26. Sakai Project. Sakai Foundation, http://www.sakaiproject.org.