ColScript a New Scripting Language for Collaborative Learning

http://dx.doi.org/10.3991/ijac.v8i4.4994

Aiman Turani
TAIBAH University, Medina, KSA

Abstract—**The collaboration script is defined as a formal way of describing the flow of activities within a collaborative learning session. Using collaboration script would encourage the production of effective and productive interactions between learners. Nevertheless, developing such script is not a trivial task. Standardization has played a major role in the expansion of instructional designs, but at the same time it limited down the flexibility of describing collaboration sessions that have complex structures. Representing a collaboration script in XML-tags format works well when scripting simple scenarios, but to describe extended scenarios it would make this scripting style very challenging and complicated. This approach causes users to avoid designing heavy weight scenarios and limits down their creativity. Relying on tools to implicitly generate such script would also limit down designers' creativity since designers can only choose from a limited set of tools and design components.**

In this research, we have defined the bases of a new collaboration scripting language, CoScript, that is able to describe collaboration learning sessions in a simple, flexible, and formal way. This scripting language has been derived based on a theoretical framework that was proposed in an earlier research. The proposed scripting language notation is close to the notation of traditional software scripting languages. This makes it easier to be learnt by instructors with basic programming skills. It has the ability to describe design's structure elements, such as sequencing, conditions, repetition, activities, activity's input /output, group formation, etc.

ColScript is basically composed of a limited set of objects and commands. The first part contains six objects (role object, group object, feedback object, collaboration tools object, time/date object and resources object), where the second part contains five essential structuring commands (input, output, loop, doactivity, groupformation).

Index Terms—**Computer Support for Collaborative Learning, Collaboration Script, Collaboration Technique, Team Collaboration.**

I. INTRODUCTION

A collaboration script has been introduced during the last decade to describe the flow of activities within collaboration sessions that are conducted in the virtual environment in a structured and formal way [1]. It describes the nature of activities, roles associated with these activities and also the environment that is needed to support such activities [2]. Careful planning of collaboration session is essential to generate productive outcomes especially within virtual environment [3]. What type of activities should be included, which roles are needed, how to proceed from one step to another are important issues when planning any session's [4]. Structuring collaborative sessions in a formal way is necessary to achieve intended objectives where in the other hand, informal collaboration does not necessarily allow participants to reach these objectives. Many reports indicate that participants suffer from various deadlocks at the start, during the process, or even toward the end of informal sessions, that for instance, include only a stated objective and a chatting tool [5].

In the other hand, designing a scripting language that is capable of describing all collaboration situations in a formal format is very challenging [6]. Defining a scripting language is not trivial and many questions need answers. What type of rules, commands, objects, etc that should be included in this language. The scripting language needs to maintain several characteristics but the two most essential ones are flexibility and simplicity. It should have the flexibility to describe a wide range of items such as role, group structures, timing, tools, resources, feedback nature, sequencing, looping, conditions, activities, etc. Simplicity is considered key issues in adopting such scripting language since managers need to play an active role in using that script.

II. COLSCRIPT LANGUAGE

In late 1990s, the first serious attempt for scripting learning designs was carried out by the Open University in the Netherlands which was called EML (Educational Modeling Language). EML was used to describe and model a learning design using a structured XML-based language. The main components of this language were: roles, activities, and environments. This language described the learning design as a flow of activities that were undertaken by roles within specific learning environment. The IMS Learning Design [7] specification was built on the top of EML as an improvement of that language. IMS LD became the fundamental standard of the current learning designs [8]. Nevertheless, following the same approach for describing heavy-weight types of collaborative sessions are not trivial and difficult to implement. XML-based languages are usually used to describe simple contents or scenarios. Trying to cope with the natural complexity within collaborative sessions' designs (conditions, iterations, notification, etc.) using XML tags would add more complexity to this type of scripting. Not many learning management systems are compliant with IMS LD versions B or C which deal with properties, conditions and notifications. Tools, such as LAMS [9], COLLAGE [10], that are used to enable instructors to implement their learning designs, still limit down their creativity since they allow instructors to select from only a limited set of tools and design components.

ColScript is a new approach of scripting teams' collaboration sessions. ColScript is built on a limited set of rules and commands similar to the other scripting languages' notations in the programming field. It consists of a limited set of programming's elements that are divided into groups, ColScript objects and ColScript commands.

III. COLSCRIPT OBJECT

There are six objects (role object, group object, feedback object, collaboration tools object, time object and resources object) that are used the ColScript scripting language. They are explained in detail in the following sections.

A. Role Object

The first step when starting a collaboration session is specifying roles [11]. Specifying the correct roles is crucial step. It indicates which participant has the right to speak at each step. For instance, in a Debate session there are usually four typical roles. They consist of two roles as debaters, an audience role, and a chairperson role. The declaration code for a debate session would look as follows:

```
Audience  as role ;
Proposer  as  role;
Opposer  as role;
Chairperson as role ;
```

B. Group Object

Proper group setting is also essential in conducting a productive collaborative learning session [12]. Grouping is a dynamic process where in many cases, more than one group setting are needed during a single session. In CoScript, the Group Object describes the group's roles and sizes. As in the previous Debate example, a group for instance, is composed from 13 participants: 2 debaters, 10 audiences, and one chairperson.

```
DebateGroup  as group = {1 Proposer,1 Opposer,10 Audience, 1 Chairperson };
```

According to this setting, the first element in the DebateGroup object would be the proposer then the opposer and so on.

To allow flexibility in group formation, group structure is based not only roles, but also could be based on other groups or even on other groups' roles. For example in a Pyramid technique, at a specific stage, two small groups are combined to form a larger group.

```
GroupL1 as group = {GroupS1, GroupS2 };
```

The second key issue that is related to ColScript Group Object is how many times to apply the formation. The formation could be applied to a single group or to multiple groups. It is more common scenario that all groups in a session keep the same settings until the end. For example in the following code, multiple-groups (g[]) within the Group Discussing technique will have 5 participants during the entire session: 4 as participants and one as a chairperson. In this case, the number of formed groups depends on the number of joining participants.

```
g[] as group = {4 participants,1 chairperson };
```

The first group would be referenced as g[1] and second would be g[2] and so on. Participant number 3 at group 4 would be referenced as g[4].3 .

The following line represents even more dynamic group formation where new large groups would be formed from combining existing two small groups (e.g. in the Pyramid technique):

```
GroupL[] as group = { GroupS[]*2 };
```

On the other hand, when breaking down large groups to the smaller ones(e.g. in All-Group-Pair technique), the code would look like this:

```
SmallGroup[] as group = {LargeGroup[]/2 };
```

The following examples represent further flexibility. A specific group could be formed based on other specific groups. For instance a new group (GG1) would be formed by only the two first large groups.

```
GG1 as group  = { GroupL[1], GroupL[2] };
```

Another example, where a new roles and groups could be formed from other group participants, is the formation of the expert group within Jigsaw technique. The expert group could be composed from the first participant of the first group, first participant of the second group, and first participant of the third group and assign them to a new role (Expert1member):

```
ExpertGroup1 as group = {GroupS1.1 as Expert1member, GroupS2.1 as Expert1member,  GroupS3.1 as Expert1member };
```

Or they could be assigned to three different roles, such as

Expert1member1, Expert1member2, and Expert1member3:

```
ExpertGroup1 as group = {GroupS1.1 as Expert1member1, GroupS2.1 as Expert1member2,
GroupS3.1 as Expert1member3 };
```

C. Feedback Object

The feedback object allows users to deliver their inputs responding to a specific task. To cover the wide range of interaction types, the feedback object should be flexible and represented in many formats. For instance, a feedback could be close or open. In case of close it might contain single, double or more feedback's option. In addition, the representation of multi-option type could have many forms such as multiple choices, lists, etc.

```
op1  as feedback.option("yes", "no");
op2  as feedback.option("1"," 2","3","4");
li3  as   feedback.list(1"," 2","3","4");
```

In case of the open type, the learner's feedback could be represented in the form of text, number (e.g. rating), or even a file.

```
txt4 as feedback.inputtext;
txt5 as feedback.inputnumber;
file1 as feedback.inputfile;
```

Feedback could be a single feedback coming from a specific learner during the session or multiple feedbacks coming from many learners. The below line of code declares f1 variable as a file type feedback. It will be assigned to an uploaded file, by the instructor for instance, during the session's runtime.

```
f1 as feedback.inputfile;
```

For feedbacks that are coming from multiple learners at a same time, an array type could be used to hold their inputs.

```
textAnswers[] as feedback.inputtext;
ideasRating[] as feedback.inputnumber;
```

D. CollaborationTools Object

This object represents a set of software tools that could be used during mini-activities. Each tool facilitates a specific activity during the runtime. For instance, a text chat tool could be used to facilitate a Group Discussion activity. The following code shows some examples of collaboration tools that could be used during the online session:

```
txtcht1 as textchat;
audicht1 as audiochat;
vid1cht as videochat;
white1 as whiteboard;
desk1 as desktopShare;
```

E. Resources Object

Resources are an essential component of any collaborative environment. Resources provide learners with the appropriate knowledge that is needed to successfully conduct a collaboration session. Regarding the Resources Object, there are four main types of resources which are:

```
doc1 as document;
img1 as image;
au1 as Audio;
vd1 as video;
```

The resources files could be assigned statically prior to runtime (d1,d2) or dynamically during the session runtime (d3).

```
d1 as document = "c:\hjh\g.x";
d2 as document  = http://www.libl\goc1.doc;
d3 as document;
```

F. Time/Date Object

The last object in this group is the Time/Date object. The Time and Date object allows session designers to specify dates and time for each activity within the collaboration session. The time and date format could be represented as in the follow examples:

```
t1 as time = 5m;
t2 as time = 00:60:00;
d1 as date = 2016:06:11:13:00:00;
```

IV. ColScript Command

In this language, there are five structuring commands that are used, groupformation, input, output, loop, and the doactivity command.

A. The groupformation Command

Groups are usually created at the start of a session where participants stay in the same group until the end of the session. However, in some cases more than one type of group formation is needed. The groupformation command allows groups to be formed according to a predefined group setting. In the following example, a small group of 4 participants and one chairperson would be formed. Then later on, all participants would be dispatched and join another group setting that is formed based on pairs;

```
Audience as role;
Chairperson as role;
Participant as role;

g[]  as group = {5 Audience, 1 Chairperson};
gp[]  as group = {2 Participant};

groupformation (g);
. . .
groupformation (gp);
```

B. The input Command

Simply, the input command allows participants to send their feedbacks during a session. This command includes an instruction element, which is an optional element, which is used to guide the participant and to clarify his task. The second element specifies the role/group who will send his/their feedback/s. The third element is the feedback object, which represents the feedback's type and its representation. Finally the last element is the exit condition, which specifies how and when to move out of this step. The exit condition could be based on a specific duration, certain role permission, the submission completion, etc. The input command syntax looks as follows:

```
input(Instruction, Recipient, Feedback Object, Exit Condition);
```

To clarify this statement more, a couple of examples are listed below. In the first example, the members of g group would be asked to input their responses in simple text format within 5 minutes. The result of the input statement type (text) should match the feedback object tv type. Note also that the result variable tv is declared as an array since there would be multiple responses coming from group's participants;

```
tx[]  as feedback.inputtext;
tx = input ("what is your opinion in ….", feed-
back.inputtext, g ,time = 5 min );
```

In another example, the group's participants are asked to vote on a certain issue;

```
ff [] as feedback.inputnumber;
ff = input ("enter scale from 1 to 100  for this .. ", feed-
back.inputnumber, g, time = 1 min)};
```

C. *The output Command*

The output command is used to direct or ask all learners or certain participant to perform a certain task. The structure of this statement is similar to the input statement. The output syntax looks as follows:

```
output (Message, Recipient, Exit Condition);
```

The following first line of code would ask only learners with p1 role to read a certain document for specific time duration. In the second line the message will appear for all group members for an unlimited period of time. The + sign indicates that this message will appear along with the old massage and will not replace it. In the third line, all g members would see the same instruction message that would disappear after 1 minute or after the facilitator decides to move to next step or after all participants had hit the finish button;

```
output  (" read this doc ….."+ doc , p1,    time= t1);
output  +(" the outcome of the dissuasion is ….." , g,  );
output  (" think about this idea by yourself ….." , g, time=
1 min or facilitator =  "finish" or g = "finish");
```

D. *The loop commands*

For allowing iterative tasks to be performed during a session, a loop command is used. The loop command is used to perform simple repetitions for a certain count or a certain condition or when covering all array's elements. The general syntax looks like this:

```
loop  (counter or condition or array as index )
{
tasks…
}
```

An example for simple count repetition of three times would be:

```
loop (counter =3)
```

Another example is a repetition until the facilitator decides to finish that process:

```
loop (facilitator ="finish")
```

The array as index statement allows for more flexible repetition depending on the array size, such as, presenting feedbacks that were sent by multiple participants, asking participants in a group to sequentially present their thoughts one by one, etc.

E. *The doactivity Command*

The doactivity command is a core command that is used to assign collaborative tasks for roles and groups.

The general syntax of this command is as follows:

```
doactivity (Instruction, From, To, Collaboration Tool,
Exit Condition)
```

The Instruction element is used to explain the task. The From element is used to specify the actor of this task. The To element is used to indicate in front of whom this task will be performed (Audience). The Collaboration Tool element indicates what type of collaboration tool is needed for this task. The Exit Condition element specifies the exit condition of this task, such as a specific time.

```
doactivity ("do….", proposer, g, aud1,  time = t1 or pro-
poser = "finish" );
```

In the above statement, learners with the role of proposer would deliver their arguments in front of their group g using an audio chat tool for a period of t1 time or when he feels he has finished and hit the finish button.

```
doactivity ("discuss this topic " , g, g,   audio1 ,  time =
t1);
```

In the above statement, all group members within g group would discuss a certain topic in front of each others for a specific period of time.

```
doactivity ("your turn to discuss this idea ", g[].1 , g,
textchat1 , time = t1);
```

In the above statement, only the first participant in the group g would be allowed to speak at that step.

V. ColScript Example

In this section, the ColScript notation is used to describe various flows of activities. Two common collaborative techniques were chosen (Debate and Group Brainstorming) due to their popularity and simplicity.

The Debate technique is used to clarify a controversial issue. The main steps of this technique are listed as follows:

1. The chairperson provide a brief background about the issue.
2. The proposer presents his argument for a specified time or when he finishes.
3. The opposer opposes that argument for a specified time or when he finishes.
4. The above two steps are repeated for another two rounds.
5. Audience participate in the discussion floor for a specific time or when the chairperson decide to move to the next step.
6. The proposer summarizes his argument.

7. The opposer summarizes his argument.
8. All participants are asked for their vote.
9. The final count of the results is displayed for all.

ColScript for the debate technique:

```
Audience  as role ;
Proposer  as  role;
Opposer  as role;
Chairperson as role ;
DebateGroup[]  as group = {1 Proposer,1 Opposer,5
Audience, 1 Chairperson };
groupformation (DebateGroup);
t1 as time = 5m;
textchat1 as textchat;
audchat as audiochat;
doactivity (“provide a brief background about the issue”,
Chairperson,   DebateGroup, audchat,  time =  t1 or
Chairperson = “finished” );
loop  (counter = 3 or Chairperson = “finished)
{
        doactivity (“propose your motion”, Proposer,
        DebateGroup, audchat,  time =  t1 or Proposer =
        “finished” );

        doactivity (“oppose the motion”, Opposer,
        DebateGroup, audchat,  time =  t1 or Opposer =
        “finished” );
}
doactivity (“all can participate in the discussion”, De-
bateGroup,   DebateGroup, audchat  textchat1,  time =
            t1 or Chairperson = “finished” );
doactivity (“summarize  your motion”, Proposer,   De-
bateGroup, audchat ,  time =  t1 or Proposer =
            “finished” );
doactivity (“summarize your opposition”, Opposer,
DebateGroup, audchat,  time =  t1 or Opposer =
            “finished” );
re[] as feedback.option("yes", "no");
re = input ("are you with this argument ", DebateGroup,
feedback .option("yes", "no") ,time =t1);
output  ("the result  for the voting ….."+  COUNT(re) ,
DebateGroup, Chairperson = “finished );
```

The Group Brainstorming technique is usually used for creating ideas or problem solutions, and then choosing the best idea or solution among them.

The session’s procedure:

1. A brief background about the problem is given to all.
2. Participants start posting ideas for a certain period of time or when the chairperson decides to move on or when all participants have posted their ideas.
3. Participants discuss each posted ideas one by one for clarification and evaluation.
4. Participants are asked to assign a mark for each idea.
5. All ideas with their marks are shown to the groups.

ColScript for the Group Brainstorming technique:

```
Participant  as role ;
Chairperson as role ;
NomGroup[]  as group = {4 Participant, 1 Chairperson };
groupFormation (NomGroup);
t1 as time = 5m;
t2 as time = 30 sec;
audchat as audiochat;
output  (" how to solve  this problem ….." , NomGroup,
,time = t1);
tv[] as feedback.inputtext
tv = input (" post your ideas ", NomGroup, feed-
back.inputtext , time = t1 or Chairperson = “finish” or
NomGroup= “finish”);

loop   (tv as i)
  {
        doactivity (discuss this idea" + tv[i], NomGroup,
        NomGroup , audchat,  time =t1 );
  }
ff[][] as feedback.inputnumber;
foreach   (tv as i)
  {
         ff[i][] = input ("enter scale from 1 to 100  for
        this “ + tv[i] , NomGroup,  feed-
back.inputnumber, time = t2);
  }
foreach   (tv as i)
  {
        output  +("the result  ….." + tv[i] + AVR(ff[i]) ,
        NomGroup, );
  }
```

VI. CONCLSION

In this paper we have proposed a new scripting language, CoScript, that formulates the basic notations and rules of defining a collaboration scripting language. This language is flexible enough to describe a wide range of different types of collaboration learning sessions and at the same time relatively simple to learn and use. For flexibility, this scripting language has followed a different approach from previous attempts of using structured XML-based languages. It has followed other scripting languages’ styles within software programming field. The structured XML-based language could be useful in describing simple sessions’ scenarios but when it comes to describe complex designs (dynamic group formation, conditions, notification, etc.), it then becomes very complicated.

We have proposed a limited set of 10 notation components that were used to compose ColScript language. The language components were divided into parts: CoScrip objects and ColScript command. The ColScript objects were: role object, group object, feedback object, collaboration tools object, time/date object and resources object where the structuring commands were: groupformation, input, output , loop, and the doactivity command.

As a proof of concepts, two scripting examples were listed to describe the flow of two different collaborative techniques using ColScript notation. In the future work, we will develop a ColScript application framework to implement such language. Instructional designers would be able use this framework not only for writing their collaboration scripts but also to develop a run-time environment to processes and support the implementation such scripts.

Acknowledgment

We would like to owe thanks to TAIBAH University, College of Computer Science and Engineering (CCSE), KSA for supporting this research.

References

[1] Dillenbourg, P. , Sanna J. , and Frank F. "The evolution of research on computer-supported collaborative learning," *Technology-enhanced learning*, pp. 3-19. 2009. http://dx.doi.org/10.1007/978-1-4020-9827-7_1

[2] Asensio, J. I., Dimitriadis, Y. A., Heredia, M., Martinez, A., Alvarez, F. J., Blasco, M. T. & Osuna, C. A. ,"Collaborative Learning Patterns: Assisting the Development of Component-Based CSCL Applications, " *12th Euromicro Conference on Parallel, Distributed and Network-based*, pp. 218-224, 2004. http://dx.doi.org/10.1109/empdp.2004.1271448

[3] Baumgartner, P., & Bergner, I. "Categorization of virtual learning activities," *In Learning Objects & Reusability of Content, Proceedings of the International Workshop ICL2003, Villach/Austria*, pp. 24-26, 2003.

[4] Goodyear, P., Avgeriou, P., Baggetun, R., Bartoluzzi, S., Retalis, S., Ronteltap, F. and Rusman, E. " Towards a Pattern Language for Networked Learning," *Proceeding of the 2004 Networked Learning, Lancaster, UK*, pp.5-7. 2004.

[5] Weinberger, A. "Scripting argumentative knowledge construction in computer-supported learning environments," *Scripting computer-supported collaborative learning*, pp. 191-211., 2007. http://dx.doi.org/10.1007/978-0-387-36949-5_12

[6] Leshed, G. "CoScripter: automating & sharing how-to knowledge in the enterprise," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1719-1728.,2008. http://dx.doi.org/10.1145/1357054.1357323

[7] IMS LD. IMS Learning Design Information Model, available at: http://www.imsglobal.org/learningdesign/ldv1p0/imsld_infov1p0.html (accessed May 2015).

[8] Neumann, S., Klebl, M., Griffiths, D., Hernández-Leo, D., De la Fuente-Valentin, L., Hummel, H., & Oberhuemer, P. "Report of the results of an IMS learning design expert workshop," 2009.

[9] LAMS. The Learning Activity Management System, available at: www.lamsinternational.com (accessed May 2015).

[10] COLLAGE. Collaborative learning design editor, available at: http://www.gsic.uva.es/collage/ (accessed May 2015).

[11] Dillenbourg, P. "Over-scripting CSCL: The risks of blending collaborative learning with instructional design," *Three worlds of CSCL. Can we support CSCL?*, pp. 61-91., 2002.

[12] Mcgrath, J. "Groups: Interaction and performance," *Englewood Cliffs, NJ: Prentice-Hall.*, 1984.

Authors

Aiman Turani is an associate Prof., Faculty of computer science and Engineering, TAIBAH University, Medina, KSA (e-mail: aimanturani@hotmail.com).

Submitted 30 August 2015. Published as resubmitted by the author 30 November 2015.