Beyond Simple Sequencing: Sequencing of Learning Activities using Hierarchical Graphs*

Sergio Gutiérrez
sergut@it.uc3m.es

Rosa María García Rioja
rgrioja@it.uc3m.es

Abelardo Pardo
abel@it.uc3m.es

Carlos Delgado Kloos
cdk@it.uc3m.es

Department of Telematic Engineering
Carlos III University of Madrid
Avenida Universidad 30
28911, Leganés (Madrid), Spain

Abstract

One of the main challenges when developing Elearning systems is the capability to adapt the learning experience to different users. This adaptability requires a flexible scheme for sequencing the learning material to different students. Content sequencing has been recently included in standardization efforts such as SCORM or IMS, namely IMS Simple Sequencing. But, is this sequencing scheme generic enough to support a wide variety of strategies? Is there a need for a “non-simple sequencing”? This document presents a procedure to translate a powerful sequencing mechanism based on hierarchical graphs specifying arbitrary transitions between learning units into the IMS Simple Sequencing standard. It can be seen that even though the translation can be performed correctly, several aspects have a non-trivial translation, suggesting areas for further exploration.

Key Words
IMS, content sequencing, hierarchical graphs

1 Introduction

Content sequencing is a crucial part of any learning activity. The advent of information technology has provided the right environment to explore sequencing techniques ranging from manual to fully automated. Also, there is an increasing effort to achieve what is called adaptation of the learning experience by which the learning context is specifically created or adapted for each student. This adaptation not only refers to the material seen by the student, but also to the sequence in which it is presented. Through adaptation, the student is given the right type of material in the right order to maximize the efficiency of the learning experience.

Parallel to this development, in recent years Elearning platforms have been widely adopted both in academic institutions and corporations. With such a large user community, it appears the need for common formats by which educational material can be reused by different tools. Standardization efforts such as SCORM [5] (Sharable Content Object Reference Model) or IMS Global Consortium [1] have been under way for the last few years. SCORM provides a collection of standards and extensions and a recommended practice of how to use them. Regarding sequencing, SCORM (starting from version 1.3) includes the standard IMS Simple Sequencing (henceforth IMSSS).

IMSSS defines a model based on an enumeration of sequencing and behavior rules. The content is organized as a hierarchy with multiple objectives defined per learning object [10]. In this document an algorithm is presented to translate a generic hierarchical graph with transition rules between learning objects into the hierarchical structure required by IMSSS.

Even though the translation can be performed correctly, several aspects mainly due to the difference in the underlying models have non-trivial translations, suggesting areas in which the model can be extended to facilitate a more powerful formalism.

The rest of the paper is organized as follows. Section 2 presents relevant work in the area of content sequencing and standards. In Section 3 the concept of hierarchical graphs is described. Section 4 describes the algorithm proposed to translate hierarchical graphs to a tree structure in IMSSS. This translation is illustrated in Section 5 with a simple example. Conclusions and future work is discussed in Section 6.

2 Related work

Content sequencing has a direct impact on the effectiveness of the learning process, thus it is present in several research areas. In the context of web-based education, content sequencing must be carefully designed to avoid the effect of “being lost in cyberspace” [2, 3]. The content to be presented to the students must be selected to adapt to the web learning environment.

Also, content sequencing is an important aspect of intelligent tutoring systems, in which this task is performed...
This document presents a procedure to translate a powerful sequencing mechanism based on hierarchical graphs specifying arbitrary transitions between learning units into the IMS Simple Sequencing standard. The purpose of this procedure is to highlight how powerful this standard is when trying to specify different sequencing strategies.

3 Hierarchical graphs

Hierarchical graphs specify how to sequence learning activities. They are powerful enough to allow arbitrary sequencing in a simple and intuitive manner, yet they can cope with big amounts of activities due to their inherent hierarchy. This hierarchy allows to store small amounts of connected activities (a plain graph) in nodes that are part of a higher level organization (another plain graph) with arbitrary nesting. They were presented in [11], where they are used to sequence parametric exercises [9].

3.1 Plain transition graph

In the following definition, a learning unit is any content to be delivered to the user of an Elearning platform. A plain graph is defined as follows:

Definition 1 A plain transition graph $G$ is a tuple $(V, E)$ where $V$ is a set of nodes each of them a learning unit and $E$ is a set of directed edges connecting nodes in $V$.

Definition 2 A condition $c$ specifies a boolean expression, either a simple one or a logic composition of simpler ones. Attribute values are divided into two types: strings and integers. Operators allowed for integer comparison are $=$, $<$, $\le$, $>$, $\ge$. Strings can only be checked for equality. The allowed boolean connectives are $!$, $\&$, $\|$ for negation, conjunction and disjunction respectively.

Definition 3 An edge $e \in E$ is a tuple $(v_1, v_2, c)$ where $v_1, v_2 \in V$. When condition $c$ evaluates to true, the corresponding transition is suitable to be taken. If several outcomes are suitable to be taken, one of them is selected non-deterministically.

Figure 1 illustrates an example of a plain graph. Nodes $A_i$ are learning units: exercises, solutions, explanations, etc. An example of condition is $(\text{attempts} < 3) \& (\text{correct answers} > 5)$.

At all times the system has a current state $A_c$ which represents the last activity or learning unit delivered to the user. When the unit has been finished, the next unit is selected according to the conditions evaluating to true in the coming edges. Edges may contain assignments, so that when a particular edge is followed, those assignments make changes in the environment.

At this point it can be seen that given a set of activities and a transition graph, the effectiveness of the system...
is captured in how transitions are enabled and how each activity modifies attribute values (thus collecting user data) altering the traversing of the graph. It is assumed that these functionalities created by a tutor or designer.

It should be noted that the proposed framework is generic enough to encompass a wide range of sequencing techniques. On one end linear sequencing of a set of conventional activities is implemented by a set of units with no parameters and a transition function returning always the next activity. At the other end, a non-trivial set of highly customizable activities are interconnected through a large number of edges labeled with conditions referring to different aspects such as scores, solution time, level of expertise, etc.

Furthermore, learning content is usually well structured. A common way of organizing educational material is through a hierarchy. A course is usually composed of a set of blocks each of them suitable to be covered over one or more sessions.

The combination of these two facts leads to the idea of hierarchical graphs, which has already been proposed and successfully used with in other scientific fields [6]. The main idea is to consider smaller graphs defining sequencing of small sets of learning units as nodes of another graph (which defines a sequencing itself) of a higher level of abstraction. A hierarchical graph is defined recursively as follows:

**Definition 4** A hierarchical transition graph $G = (V, E, v_i, v_o)$ is a tuple where elements in $V$ are either learning units or transition graphs, $E$ is a set of edges, $v_i \in V$ is its input node and $v_o \in V$ is its output node.

With this new definition, a hierarchical graph is a set of learning units and subgraphs connected among them by a set of edges. Two of its nodes are the input and output nodes. The input node is the entry point from a higher subgraph. The output node provides the transitions into nodes of a different node in a higher subgraph. There is one and only one input and one output node in each plain subgraph.

This definition becomes more intuitive when the hierarchical transition function is defined. This definition given algorithmically in Figure 2.

An example of a hierarchical graph is given in Figure 3. The hierarchy contains two levels with nodes labeled starting with “A” and “B” respectively. Input nodes are denoted by a white incoming arrow, and output nodes are denoted by a cross at the bottom.
4 Translation Algorithm

One of the main difficulties to express hierarchical graphs in terms of IMSSS is that IMSSS is based on a tree, which must be traversed in a canonical pre-order sequence. This aspect makes it difficult to define arbitrary sequences. Furthermore, information about objectives in IMSSS only indicates if they have been satisfied or not, but does not allow for more precise information (i.e. grade of satisfaction). This restriction makes it more difficult to translate a system where transitions between states is done according to the values of previous answers not just if these answers were correct.

The algorithm is explained in two steps. First, it is shown how a plain graph (with no hierarchy) can be translated into IMSSS. Then, a correspondence and transitions between the levels in a hierarchical graph and an IMSSS tree is defined.

4.1 Plain Graph Translation

The algorithm to obtain an IMSSS tree from an arbitrary plain graph proceeds in the following steps:

1. Given A, the set of nodes of a graph, a tree is created with one root and as many leaves as nodes are in A: for each vertex \( v_l \) there is a leaf \( l_i \). The order of the leaves is irrelevant as long as the input node is situated first and the output node is situated last.

2. Given an edge \( \gamma \) connecting nodes \( v_1 \) and \( v_2 \), and given a condition \( c \) associated with that edge, a pre-condition is added to every leaf between \( l_1 \) and \( l_2 \) in the tree, so that they are “disabled” when \( l_1 \) has been delivered and condition \( c \) is true. This way, the pre-order transversal reaches \( l_2 \) after \( l_1 \) when \( c \) is satisfied, no matter the leaves in between.

3. Each leaf \( l_i \) in the tree is examined according to its originating node \( v_i \) and outgoing edges.
   - If all edges go to nodes the corresponding leaves of which are after the current leaf, then a post-condition is added with a continue event.
   - Similarly, if all edges aim to “preceding” nodes, a post-condition is added with a previous event.
   - In the general case, if some edges go “forward” and some “edges” go backward, two post-conditions are added:
     - The first generates a continue event when any of the “forward” conditions (those associated to edges going “forward”) is true.
     - The second generates a previous event when any of the “backward” conditions is true.

4. For each assignment A (in the form name = value) that appears in an edge, an objective is created that notifies if this assignment has happened.

After the algorithm completes, a cluster of activities is produced such that the pre-order traversal according to IMSSS rules is equivalent to that of the original plain graph.

4.2 Hierarchical Graph Translation

The translation algorithm applied to a hierarchical graph makes use of the previous algorithm for plain graphs.

The first step is to translate all plain graphs as explained in Section 4.1. A set of disjoint plain graphs is obtained, with internal transitions defined in terms of several conditions (graphs semantics) and objectives (IMSSS semantics).

IMSSS arranges the learning activities as an activity tree. This tree does not need to be balanced, nor its branches need to be of equal size. As in every tree, there is an implicit containment hierarchy. It is this implicit hierarchy the one used to reflect hierarchical graphs.

Special care must be taken with sequencing events such as “previous” or “continue” (generated by the post-conditions of a node, if any) because the canonical pre-order traversal of a IMSSS tree involves a sequence of activities that mixes different layers, which in principle is not related to the layer-based sequencing behavior of hierarchical graphs. The translation process follows these steps:

1. The set of activities corresponding to plain graphs that are inside a higher level node (such as \( A1 \rightarrow A4 \) inside \( B3 \) in Figure 3) are set to be children of the activity corresponding to that node. This is repeated until there is only one IMSSS tree with all the activities. A virtual node with no content is added at the top to have a single-rooted tree if required.

2. Post-conditions of a cluster activity (one with “children” activities) have to be moved to the activity corresponding to its output child node (the last node visited under it). This way, the post-conditions are evaluated after all the children nodes have been delivered.

   If these post-conditions involve the generation of “previous” events, a pre-condition must be added to itself, its children, and children of the preceding activity. This pre-condition disables the activities when the condition present in the moved post-condition is true. When the “previous” event is generated, the student is directed to the previous sibling (or another one before, depending on the corresponding edge) of the node skipping the rest of activities that follow the reversed pre-order traversal of the tree.

   This step is applied for every cluster activity bottom-up until the whole tree has been created.

5 Example

In this section the translation procedure is illustrated with examples for plain and hierarchical graphs. For the sake of
simplicity, the complexity of these examples has been kept to a minimum.

The graph in Figure 1 has four activities. These are actual learning activities, they do not represent lower level of a hierarchy (they do not have “children”).

In the following, exercise is used as an equivalent for learning activity consisting of an exercise.

The graph represents a set of four exercises of a unit in a Computer Science course. Activity A1 represents a set of exercises about different numerical bases; they intend to check that a student has clearly understood these concepts. Activity A2 consists of a theoretical explanation about numerical bases. A3 is an exercise of moderate difficulty involving bus sizes and machine instruction lengths. A4 portrays the final exercise of the unit, a complex one involving bus sizes, word lengths, different instructions of a simple microprocessor and additional elements like a stack.

The student starts by solving A1. Most students will solve correctly the exercise and will move forward to the next activity, A3. Those students who fail exercise A1 are assumed to have difficulties with numerical bases and therefore are redirected to the theoretical activity A2. After properly completing exercise A3, students move to A4. Also, if the system stores information about the student performance it may infer that there is no need for an intermediate exercise A3 and it delivers activity A4 right after A1.

In the following discussion, lower-case letters denote the activities in the IMSSS tree and the nodes in the graph are denoted by upper-case letters.

5.1 Plain Graph Translation

The nodes are mapped to a sequence of activities, in which a1 is the first (A1 is input node) and a4 the last (A4 is output node). For this example, the sequence a1 − a3 − a2 − a4 is assumed.

After the sequence has been decided pre-conditions are added to each activity, so that the behavior determined by the conditions associated to each outgoing edge is expressed correctly. For example, when A1 is successfully completed and condition c6 is satisfied, edge A4 is traversed. This results in pre-conditions in both activities a3 and a2 to disable them when the last delivered activity was a1 and the set objectives encoding c6 is satisfied. These two logical conditions are mapped to two rule conditions, and the edge condition is mapped to rule condition referenced objective following IMSSS conventions SM.2.2.1 and SM.2.2.2 (see [4]). An analogous procedure is performed for all the remaining edges. In the end, the resulting set of disabling pre-conditions is:

\[ a3: \text{last-activity-was-A1 AND advanced\_student}, \text{last-activity-was-A2, failed-at-A1 AND last-activity-was-A1}. \]

Additional forward and backward post-conditions are needed as well to control the traversal of the tree. Thus, a post-condition is added to a4 so that if objectives representing c3 are satisfied a “previous” event is generated. This, combined with the corresponding disabling pre-condition in a2, leads to a3, the behavior specified in the original graph. If not, this is the final activity. A similar reasoning leads to post-conditions in a1 and a3 that generate a “continue” event in any case (all edges go “forward”) and in a2 that generate a “previous” event in any case (the only edge leads to a preceding activity).

After the algorithm has finished the following set of post-conditions is created:

\[ a1: \text{always continue}. \]
\[ a2: \text{always continue}. \]
\[ a3: \text{always previous}. \]
\[ a4: \text{IF objectives for c3 THEN previous}. \]

All IMSSS objectives used during the process need to be derived from the conditions in the graph prior to the translation process.

5.2 Hierarchical Translation

The translation of hierarchy is illustrated with the graph in Figure 3. Two levels can be observed. The higher one depicts a graph representing a simple computer architecture course. Node B1 represents a preliminary unit about computer architecture, B2 is a unit about memories, B3 relates to microprocessors and B4 is the last unit about new tendencies in distributed architectures. Only one graph of a lower level is presented and it corresponds to B3.

After the first unit is completed, either c11 or c12 (or both) are true, depending on the system knowledge about the student. If the student has a good foundation about memories the unit B2 is sequenced, otherwise, she is directed to B3. From B2 (memories) the student can follow one of two paths: either B3 and then B4; or directly B4 if B3 is known (the way to acquire this knowledge about the student is system-dependent). A similar reasoning applies to B3. Notice the flexibility that the graph allows with only four different units.

The mapping process for plain graphs is followed at every level of the hierarchy. This leads to the following set of pre-conditions and post-conditions in the activities b1 to b4:

Pre-conditions (all disabling):
\[ b2: \text{last-activity-was-B1 AND NOT probably-memories-known}. \]
\[ b3: \text{last-activity-was-B2 AND micro-known}. \]

Post-conditions:
\[ b1: \text{always continue}. \]
\[ b2: \text{always continue}. \]
\[ b3: \text{IF memories-known THEN continue, IF NOT memories-known THEN previous}. \]

1 The term disabling pre-condition is used henceforth for pre-conditions the action of which is always “disabled.”
Now, the mappings of for all plain graphs must be rearranged to express the hierarchy of the whole graph. In this case, activities $a_1$ to $a_4$ are created to be “children” of activity $b_3$, and the same is done for the children of the other nodes. If higher levels of hierarchy existed, this steps are repeated for each level.

Once the hierarchy is set, several changes must be done in order to preserve the correct behaviour of the graph when the tree is traversed and a change of “layer” happens. These changes affect directly post-conditions of higher layers and in this case involve the creation of additional pre-conditions.

In this example, post-conditions of activities $b_1$ and $b_2$ are very simple. They are moved to the activity corresponding to the output node bellow them (not shown in the example). But the interesting point here is activity $b_3$: edges from node $B_3$ go to nodes $B_2$ and $B_4$, so post-conditions generate both “continue” and “previous” events according to the result of a condition.

The case of edges going to $B_2$ if objective “memories-known” is satisfied, the student has to be directed to $b_4$ and its children, so that first post-condition is directly moved to $a_4$, activity corresponding to output child node of $B_3$. The case case of edges going to $B_4$ is a bit more complex: when objective “memories-known” is not satisfied, a “previous” event needs to be generated (so the traversal direction leads to $b_2$), as well as many pre-conditions. In the example, the second post-condition is moved to $a_4$, and a pre-condition is added to $a_3$, $a_2$, $a_1$, and all the children of $b_2$, so that when objective “memories-known” is not satisfied, they are “disabled”. This way, the “previous” after $a_4$ (thus, after the end of $b_3$) leads to $b_2$, as required.

6 Conclusions and future work

A flexible and powerful way to sequence educational contents based on hierarchical graphs, and an algorithm to express it in terms of the well-accepted IMSSS standard has been presented.

The use of transition graphs allows to define arbitrary sequencings of learning units, according to the tutor’s knowledge and the user’s previous history throughout the system. Learning units are connected by edges that have associated conditions. These conditions determine which unit is to be delivered at any time, thus defining a sequencing of information that can take into account the past history of the user. Different levels of hierarchy allow to develop sequencings for large amount of units easily, defining different levels of abstraction.

The possibility of expressing hierarchical graphs with IMSSS semantics allow Elearning applications developers to use this paradigm as an intuitive formalism to sequence learning content without limiting the possibility of interchanging information between different applications.

But the process has been proved to be non-trivial, and the high number of pre-conditions and post-conditions needed could produce an explosive growth of the resulting XML files used for sequencing. This could be a symptom that the paradigm used by the IMSSS standard is too restrictive to allow arbitrary sequencing.

This work has inspired several lines to explore in order to obtain a more compact representation to capture a wider variety of content sequencing techniques.

References