Sequencing Parametric Exercises for an Operating System Course

Pilar Prieto Linillos, Sergio Gutiérrez, Abelardo Pardo, and Carlos Delgado Kloos

Department of Telematic Engineering
Carlos III University of Madrid, Spain
{pilar_pl,sergut,abel,cdk}@it.uc3m.es

Abstract. An adaptive tutoring system for an Operating System course is presented. The architecture, based on sequencing graphs, that supports an adaptive sequencing of the learning units is described. The content structure is presented as well. The system is now in use in regular university courses and results of this experience will be published in the future.

1 Introduction

Web based education (henceforth WBE) has seen in recent years a significant increase in both in its functionality as well as possible scenarios. E-learning systems are now present in an ever growing number of companies as well as educational institutions of all levels. After a first stage in which these systems offered mainly content management and course management capabilities, systems now offer solutions that cover pedagogical aspects such as activity sequencing.

When users of an e-learning platform are simply given access to a set of documents for each course, there is a high risk of being "lost in cyberspace" [1]. As important as having access to the proper documents, course activities and how they are organized within a course have a direct impact on the overall effect of the learning experience. Adaptive Hypermedia is a research area that focus on how hypermedia can be changed according to the user needs. When applied to e-learning, these techniques are generally known as "personalized learning" or "adaptive educational hypermedia". The idea is to customize learning material and activities and provided a personal environment for each learning [2].

This paper presents how a sequence of parametric exercises has been deployed in the context of a course on Operating Systems. Parametric exercises are those with content suitable to be instantiated an unlimited number of times with different data [3]. A set of exercises are initially designed and hierarchically organized by topics. A transition structure similar to a state machine is defined.

The presented framework allowed the teaching staff to organize a significant set of exercises from different operating system topics such as process scheduling, memory management, disk organization, and file system techniques. As a result a web based tutor offers the students a personalized exercise sequence based on the previously given answers. The system has been implemented and

Please use the following format when citing this chapter:
Linillos, Pilar Prieto, Santos, Sergio Gutierrez, Pardo, Abelardo, Kloos, Carlos Delgado, 2006, in IFIP International Federation for Information Processing, Volume 204,Artificial Intelligence Applications and Innovations, eds. Maglogiannis, I., Karpouzis, K., Bramer, M., (Boston: Springer), pp. 450–458

tested in a reduced environment and is now being used in regular university courses.

The rest of the document is organized as follows. Section 2 presents related work in the area of tutoring systems and e-learning content sequencing. Section 3 describes how the material is organized to cope with its complexity as well as the conditions to describe the transitions. Design of the tutor on operating systems is described in 4. An example of exercises covering a concrete topic of the course is shown in 5. The document terminates with a brief outline of future work in 6

2 Related Work

There is a wide variety of strategies proposed for sequencing educational material even before the appearance of e-learning platforms. In the area of computer assisted learning, intelligent tutoring systems [4] typically include a "tutoring model" that decides which information to be shown to the user. Techniques such as bayesian networks or neural networks (to mention a few) are used to deduce such information. The common characteristic of these systems is that designers do not specify directly this sequence but an algorithm to compute it.

The ideas included in this paper asume that a the designer specifies such sequence (or set of possible sequences). In this scenario several languages have been proposed to define these sequences. They are usually referred to as "Educational Modelling Languages" or EMLs (see [5] for a survey). Other techniques use already existing languages. As an example, in [6] UML (Unified Modelling Language) is used to define the sequences emphasizing in the use of ontologies to guide the student.

Two more initiatives that seem to capture the interest of the e-learning comunity are Simple Sequencing [7] and Learning Design [8]. Both of them are specifications proposed by the IMS Global Consortium and have become part of SCORM [9]. Simple Sequencing assumes the material is organized as a tree and provides conditions to modify an in-order traversal of the nodes. Learning Design tackles a wider problem because it allows the possibility of specifying arbitrarily complex activities such as discussions, collaborative tasks, etc.

The main problem with these techniques is that require the designer to learn an additional notation to formally describe a learning experience. The proposed approach can be considered as a tradeoff between expressive power and simplicity. The goal is to capture the experience teaching staff has about how material should be sequenced with the most intuitive formalism possible, a graph.

In [10], Collet et al. propose using a graph to describe a sequence of problems in a way similar to the one propose in this document. Transitions are initially defined by a user and later modified depending on the probabilities associated with the transtions. The algorithm used to modify these probabilities follows an ant colony optimization strategy.

In the proposed approach, the designer is in charge of providing a transition structure with a set of conditions that consider different scenarios when solving the exercises. Instead of dinamically adjusting the graph structure, the conditions attached to the transitions allow for a large number of possible paths along the graphs. The main challenge when deploying this paradigm has been taking a concrete course on operating systems and designing the transition graph with concrete exercises.

Sequencing graphs, presented in the next section, specify how to sequence learning activities in our system. They are powerful enough to allow arbitrary sequencing in a simple and intuitive manner, yet they can cope with big amounts of activities without becoming unmanageable 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 organisation (another plain graph, but with graphs inside).

3 Sequencing Graphs

In our system, Sequencing Graphs define the sequencing of learning units. In the following definition, a *learning unit* is any digital learning content to be delivered to the user of an elearning platform including, but not limiting to, an exercise, its solution, a page with formulae and explanatory text, a commented photograph, etc.

3.1 Plain graph

A Sequencing Graph is defined recursively. In this subsection the plain graph with no hierarchy is presented. The hierarchy is explained in 3.2. A plain graph is defined as follows:

- 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 arcs connecting nodes in V.
- An *environment* Π is a set of pairs variable-value, where information about the student and its relation to the graph can be stored. Attribute values are divided into two types: strings and integers. Changes in the environment are made after each unit is delivered (any output data is stored in it) and, more importantly, by the *actions*.
- An *action* a determines a change in the environment. This can be the addition of a new pair to the environment, the change of an existing one or the deletion of it (PUT actions and DEL actions).
- A *condition* c specifies a boolean expression. Operators allowed for integer comparison are $=$, $<$, $\leq$, $>$, $\geq$. Strings (including booleans) can only be checked for equality. The allowed boolean connectives are ! , &, | for negation, conjunction and disjunction respectively.

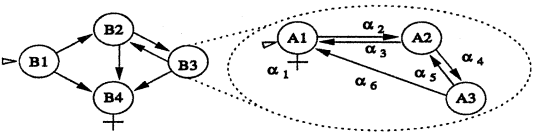

Fig. 1. Sequential graph hierarchy example.

An arc $\alpha \in E$ is a tuple (v_1, v_2, c, A) where $v_1, v_2 \in V$. When condition c evaluates to true, the corresponding transition is suitable to be taken. Should the arc be followed, the corresponding set of actions A would be executed, modifying 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. The proposed system assumes these functionalities to be created by a tutor or designer and provides a user friendly environment to develop the transition structure, allowing for a solid student supervision using the collected data.

It should be noted that the proposed architecture 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 arcs labeled with conditions referring to different aspects such as scores, solution time, level of expertise, etc.

3.2 Hierarchy. Sequencing graphs.

Although plain graphs provide a powerful and flexible mecanism to express sequencing of learning units, they may become too complex for a large number of units. In this situation defining or maintaining a large transition structure can become infeasible.

A possible solution to overcome this is the use of hierarchy. Diferent kinds of *hierarchical graphs* have been already proposed and used with good results in other scientific fields [11]. The main idea is to consider smaller graphs defining sequencing of small sets of learning units as a node of another graph (which defines a sequencing itself) of a higher level of abstraction.

A sequencing graph is thus defined recursively as follows:

A *sequencing graph* $SG = (V, E, V_i, V_o)$ is a tuple where elements in V are either learning units or sequencing graphs, E is a set of arcs, $V_i \subset V$ is its set of input nodes and $V_o \subset V$ is its set of output nodes.

With this new definition, a sequencing graph is a set of learning units and subgraphs connected among them by a set of arcs. The input nodes are the possible entry points from a higher level of hierarchy. The output node are those with arcs directed to the upper level of hierarchy.

The scope of the environment may be general (one and only Π for the whole SG) or there could be a different environment for each level (Π_l) of hierarchy, with a mechanism to copy variables from one level to the upper ones. We have selected the latter approach, in order to make the implementation easier to use, and defined a SAVE action that copies a variable from the current environment to the upper level one (see section 5 for an example).

The possible sequencings defined by a sequencing graph are very intuitive to see. Nevertheless, in [12] the traversing algorithm is formally expressed. As we have not made any change to that, we do not repeat it here for the sake of space.

An example of a sequencing graph is given in Figure 1, with two levels of hierarchy which nodes are labeled with letters A and B. Input nodes at each level are marked with a white incoming arrow. Output nodes are marked with a cross, representing an arc going to the parent level. Output nodes could have a number of outgoing arcs going to the "parent", each one with a different condition and a different set of actions, if this fits the pedagogical purpose of the graph.

4 Tutor Structure

The course considered in this work is included in the fall semester of a fourth year in Telecommunication Engineering Degree. The course material is a subset of what is usually covered on a course on this topic in most universities. The reference book is "Operating Systems Concepts" [13]. The course objective is for the students to become familiar with the types of problems present in any operating system as well as a sample of techniques used to solve them.

The first decision to design the tutor was to select those areas that would be covered. The selected concepts were: process management, cache memory, memory management, disk management and file systems. In principle, any topic included in the course was suitable to be included in the tutor. However, these topics were selected because they were identified as those that posed a greater difficulty for the students to assimilate and where exercise solving would most likely had a positive effect. Despite of being a topic more suitable for a computer architecture course, cache memory is part of the course material due to dependencies derived from the entire degree.

The design process for the tutor followed a hierarchical approach to benefit from the architecture discussed on Section 3. The first level sequencing structure is trivial. Topics are covered sequentially as explained in the course lectures. The reason why no other possibility was allowed at this level is concept dependency.

Module	Learning Units	Exercises	Documents
Process Management	Scheduling Policies	5	6
	Scheduling Policy Analysis	2	1
	Performance Analysis	2	1
Cache Memory	Memory Access Time	1	1
	Direct Mapping	5	1
	Associative Mapping	2	1
	Set Associative Mapping	4	1
Memory Management	Contiguous Assignment	2	4
	Paging. Access Time	1	1
	Paging. Address Space	3	1
	Paging. Page Tables	2	2
	Paging. Process Allocation	1	1
	Paging. Page Replacement	2	3
Disk Management	Disk Access Scheduling	2	6
File Systems	Access Permissions	2	1
	Inodes	1	1
	Sector Allocation	3	2
Total		40	34

Table 1. Learning Units, Exercises and Documents for each Module

Concepts covered in one module make use of the ones previously covered. All sequence adaptation is then contained in each module separately.

Within each module, the material was divided into "learning units". In the context of this work, a learning unit is informally defined as a set of topics containing a set of exercises and a set of documents explaining the concepts required to solve these exercises.

The learning units present in the tutor as well as the number of exercises and auxiliary documents are shown in Table 1. As it can be seen, for each module, a subset of topics were selected. Again, each topic can be arbitrarily extended to cover more learning units, but the presented selection was based on the difficulty perceived by the students as well as how feasible was to design a set of effective exercises.

5 An Example

The tutor is composed of many graphs and subgraphs. The smallest ones are composed of only two elements, with an arc connecting both: one corresponds to the question and the other one with the grading of a particular parametric exercise. The biggest ones have almost ten elements, and up to twenty arcs conecting them (figure 2 is an example).

In this section we will explain in some detail one of the graphs (the right half of figure 1), the one corresponding to the section about 'Access permission' section of the tutor.

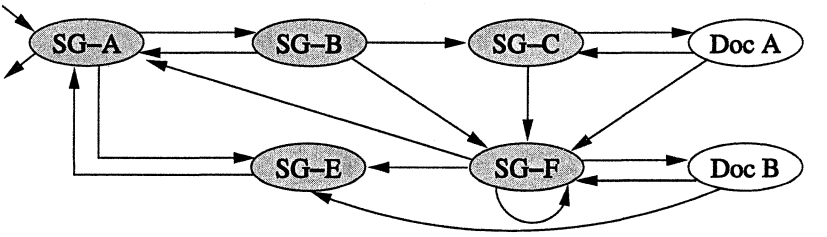

Fig. 2. Example of Graph for Learning Unit

The objective of this section of the tutor (graph) is to assess the knowledge of the student about file permissions in UNIX. First, it shows the student an exercise. If the students shows a great expertise on the matter, the section is finished. If not, another exercise of lower difficulty is delivered. If the student fails on the easy exercise, it is assumed that he knows nothing on the topic, so an auxiliary document is delivered and the student is advised to read it. More exercises are delivered then: good answers will lead to more difficult problems and bad answers will move the student in the opposite direction. The cycle is repeated until the student answers correctly all the questions of the hard exercise or has tried a number of times. At that moment, the section is finished (the sequencing goes up a level) and the tutor selects another section. The marks of the student are saved so that the tutor knows if this section should be revisited.

The graph has three nodes. Nodes A1 and A2 are sub-graphs and node A3 is an activity itself. Nodes A1 and A2 are graphs containing the exercises and node A3 links to the theory web page. Node A1 is both the only entry node and the only exit node. The complete set of conditions and actions is shown in table 2.

When the user comes to this level (graph) from an upper level of hierarchy, a new environment is set for him at an *init* phase, and it is initialised with only one new pair: *penalization_acc_perm = 0.*

The variable *penalization_acc_perm* acts like a counter to ensure that the student does not get stucked in this graph. Every time the student fails in one of the two exercises (nodes A1 and A2), condition in arcs 2 and 4 are evaluated to *true* and the arc is followed. The corresponding actions increase the value of *penalization_acc_perm* by 250, with a maximum of 1000. Conditions in arcs 1 and 6 ensure that after four or five (depending on the spacific path of the student) failed attempts, as the value of *penalization_acc_perm* will have reached 1000, the student will go up to the upper level. There he will be redirected to another part of the tutor and come back here later to cover this part.

It can be seen that variables can be put into the environment at the *init* phase or by the actions when an arc is followed. This is the case with *grade_acc_perm*, which is created and introduced when arc 1 is followed to the

Arc	Condition	Action(s)
1	all_evaluation=correct OR penalization_acc_perm=1000	[1] PUT grade_acc_perm = = 1000 - penalization_acc_perm; [2] SAVE grade_acc_perm
2	all_evaluation=incorrect AND penalization_acc_perm < 1000	[1] PUT penalization_acc_perm = = min(1000,penalization_acc_perm + 250); [2] DEL all_evaluation
3	all_evaluation=correct	[1] DEL all_evaluation
4	all_evaluation=incorrect	[1] PUT penalization_acc_perm = = min(1000,penalization_acc_perm + 250); [2] DEL all_evaluation
5	penalization_acc_perm < 1000	—
6	penalization_acc_perm=1000	—

Table 2. Conditions and actions for 'Access permission' graph

parent node. Furthermore, as this variable will be needed for the sequencing at the upper level and the level's environment is erased as the student goes up, a SAVE action is needed so its value is not lost.

The third way in which the environment can be modified is by the learning units themselves, or by lower level graphs, using SAVE actions. This is the case of variable *all_evaluation*. This variable is the result of a SAVE action from the lower level, either from node A1 or A2. As children of these nodes are exercises, they set several variables in the environment (the lower level's one). When the output arc is taken, a PUT and a SAVE action on it set this *all_evaluation* variable in this level. Its value determines whether arcs 1 or 2, or 3 or 4 must be followed. Note that, when the corresponding arc is followed, a DEL action erases *all_evaluation*. This is necessary so the result of the exercise under A1 does not affect the sequencing after A2, and viceversa.

6 Conclusions and Future Work

An application for sequencing parametric exercises about operating system topics has been presented. Using sequencing graphs to define the transitions between exercises, a web-based tutoring system has been designed that adapts the sequence of exercises to the capabilities and needs of the students. The system is now in use in regular university courses. Results of this experience will be published in the future.

There are two main lines for future work, both in relation with the sequencing graphs that the tutor is based on. First, they have been developed based on the past experience of the teachers of the target courses, in an ad-hoc fashion. Further research and application of these ideas should lead to some best practices that will make it easier to apply them by other educators, much in the way that design patterns [14] can be useful. On the other hand, a graphical editing tool is necessary to make the design process easier and faster, and it is now in development.

References

1. Brusilovsky, P.: Adaptive educational hypermedia. In: International PEG Conference. (2001) 8–12
2. Cristea, A.: Authoring of adaptive and adaptable educational hypermedia: Where are we now and where are we going? In: IASTED International Conference in Web-Based Education. (2004)
3. Brusilovsky, P., Miller, P.: Course delivery systems for the virtual university. In Tschang, F.T., Santa, T.D., eds.: Access to Knowledge: New Information Technologies and the Emergence of the Virtual University. Elsevier Science (2001) 167–206
4. Murray, T.: Authoring intelligent tutoring systems: An analysis fo the state of the art. Int. J. of Artificial Intelligence in Education **10** (1999) 98–129
5. Rawlings, A., Rosmalen, P.V., Koper, R., Rodrguez-Artacho, M., Lefrere, P.: Survey of educational modelling languages (EMLs). Technical report, CEN/ISSS WS/LT Learning Techonologies Workshop (2002)
6. Dolog, P.: Model-driven navigation design for semantic web applications with the uml-guide. Engineering Advanced Web Applications (2004)
7. IMS Global Learning Consortium: IMS Simple Sequencing specification (2003) v.1.0.
8. IMS Global Learning Consortium: IMS Learning Desing specification (2003) v.1.0.
9. Learning, A.D.: SCORM 2004 (sharable content object reference model). Specification (2004)
10. Semet, Y., Lutton, E., Collet, P.: Ant colony optimisation for e-learning: Observing the emergence of pedagogical suggestions. In: IEEE Swarm Intelligence Symposium. (2003)
11. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming **8** (1987)
12. Rioja, R.M.G., Santos, S.G., Pardo, A., Kloos, C.D.: A parametric exercise based tutoring system. In: Frontiers in Education Conference. (2003)
13. Silberschatz, A., Galvin, P.: Operating Systems Concepts. 7^{th} edn. Addison-Wesley (2004)
14. Avgeriou, P., Vogiatzis, D., Tzanavari, A., Retalis, S.: Design patterns in adaptive web-based educational systems: An overview. Advanced Technology for Learning (2004)