In: G. Davies and C. Owen (eds.) Proceedings of WebNet'2000, World Conference of the WWW and Internet, San Antonio, TX, Oct. 30 - Nov. 4, 2000, AACE, pp. 69-74.

Concept-Based Courseware Engineering for Large Scale Web-based Education

Peter Brusilovsky

Carnegie Technology Education, Carnegie Mellon University

4615 Forbes Avenue, Pittsburgh, PA 15213, USA

plb@cs.cmu.edu

This paper describes a concept-based course maintenance system that we have developed for Carnegie Technology Education. The system can check the consistency and quality of a course at any moment of its life and also assist course developers in some routine operations. The core of this system is a refined approach to indexing the course material and a set of "scripts" for performing different operations.

Introduction

One of the major functions of a modern Web-based education (WBE) system is to store and deliver over the Web *the course content* (explanations, examples, quizzes, assignments, etc.). A large variety of course *support tools* and systems [Brusilovsky & Miller 2000, in press; Robson 1999] is now available for creating, storing and delivering the course content. The problem, that we are addressing in this paper, is that the vast majority of these tools is oriented implicitly to a *single author content development model*. In this model a syllabus and all course material is developed mainly by a single author in about the same way as a single-author textbook. Single author is usually able to develop a consistent course and update once a year to time without a serious loss of consistency. The situation with the large-scale Web-based education model that is emerging now is quite different. Large-scale modern courses includes hundred to thousands of learning items that are produced by a team of developers. Through the life of the course it could be updated and restructured several times to accommodate to the needs of different audiences. The difference between single-author and large-scale Web-based courses is a very similar to a difference between small programs developed by single programmer and large programming complexes developed by a team of software developers. Producing large-scale artifacts either programs or courses requires special support. Large software systems require the use of software engineering technologies and tools. Large-scale courses require the user of courseware engineering technologies and tools.

A special kind of courseware engineering tools is concept-based consistency maintenance tools. Consistency maintenance could be performed for a course where all learning items

are extended with metadata that describes prerequisites and outcomes of every item in terms of *concepts*, *topics*, or *learning objectives*. The very idea is simple. A consistency checking mechanism can parse a course in the same order as a student and at any point check whether the "next step" is a good one. If the next step is not really good, it can report problems. For example, it can find a situation when an assessment requires knowledge that are not presented yet or, vice versa, when presented knowledge are never assessed. While these kinds of course checking seems too simple to require a special system they are absolutely necessary for any serious course developer teams such as Carnegie Technology Education, a WBE "arm" of Carnegie Mellon University. .A concept-based course maintenance system is as important for courseware engineering as a version tracking system for software engineering.

This paper describes a concept-based course maintenance system that we have developed for Carnegie Technology Education. The system can check the consistency and quality of a course at any moment of its life and also assist course developers in some routine operations. The core of this system is a refined approach to indexing the course material and a set of "scripts" for performing different operations. Next section describes the indexing part and the section after that talks about scripts. We conclude with some speculation about prospects of our work.

What do we want from indexing

There are several possible ways to index the content from very advanced and powerful to very simple. The approach we choose supports the functionality that we find essential while being still simple enough to used by course developers.

Most of the existing indexing approaches are based on prerequisite-outcome concept indexing like the one used in Piano-Tutor [Capell & Dannenberg 1993] or InterBook [Brusilovsky, Eklund & Schwarz 1998]. Plain indexing does not distinguish different types of information items and different roles in which a concept can be involved in a learning item. It also does not take into account relationships between concepts. Plain indexing has shown to be useful in simple domains or with coarse-grain level of domain modeling. All systems with plain indexing known to the author use about 50 concepts and have homogeneous learning items (lessons, chapters, or pages).

The three major extension of plain indexing approach are introductions of item types, concept roles, and links between concepts. Item types let the system distinguish several types of indexable information items. Concept roles can specify different roles of the items in regard to concepts. Both mechanisms let the course developer specify more knowledge about the content and support more powerful algorithms. Item types are simpler and easier for indexing, however, it's mainly suitable for reasonably small information items that can be easily typed. Role-based indexing approach is more

powerful – it can deal with cases where different concepts play different roles for same information item. It could be used with coarse-grain items. A few systems with exceptionally rich information space use both concept typing and role-based indexing. Inter-concept links is a serious advancement. Its major impact is a more precise student modeling, prerequisite tracking, and richer navigation. Negative side of it is hard authoring. Developing a connected concept model of a domain takes considerable time of several domain experts.

Since the major goal of indexing of CTE courses was the possibility to help course designers with developing and modifying courses, we were interested to represent as much information about items, as it is reasonable for the authors and decided to all extensions: types, roles, and links.

The core of our framework are concepts – elementary pieces of learning material. The size of a concept is not fixed and may depend of a course. We have several kinds of teaching operations in our courses – presentations, examples, assignments, and multiple-choice questions. The type of the item is a part of the index for the item. The rest of the index is formed by concept-role pairs. We use four kinds of roles (in comparison with only two in InterBook and Piano-Tutor): light prerequisite, strong prerequisite, light outcome and strong outcome. In comparison with "real" or strong prerequisites and outcomes that tells that "deep" knowledge of a concept are produced or demanded by a learning item, the light prerequisites and outcomes deal with surface knowledge about a concept. We have to introduce these four roles to accommodate the needs of real courses.

The course concepts are connected to form a heterarchy. We use one non-typed parent-child link. This link has to express the value usually expressed by "part-of" and "attribute-of" links. Creating a parent-child hierarchy without the need to type links is relatively easy The meaning of this hierarchy is simple – the knowledge of a parent concept is a sum of knowledge of child concepts plus some "integration extra". There are two major uses of this parent-child heterarchy:

- A parent concept could replace a list of all child concepts in indexing (for any role). This will make the indexing more compact and easy in use.
- When monitoring individual students, the current level of knowledge of 30 higher-level concepts provides a better view on the student progress than the state of knowledge of 300 terminal-level concepts.

The use of indexing for courseware engineering

Prerequisite checking

Prerequisite checking is the one of the key benefits of concept indexing. It is important for original course design as well as for a redesign when learning items are moved or

changed. With multiple-level indexing we are able to check prerequisites for all learning items. Prerequisite check for linear courses is performed by a sequencing engine that simulates the process of teaching with a student model. It scans learning items in the order specified by the author, updates the student model, and checks the match between the current state of the model and each following item. The following prerequisite problems could be checked:

- **Presentation prerequisites:** a presentation item can be understood because all prerequisite concepts are already presented up to the required level
- **Question prerequisites:** all concepts involved into all questions designed for a presentation page are learned at least up to the advanced level when the page is completed.
- **Example prerequisites:** all concepts involved into an example are learned to the required level right in the section where an example is presented or before; strong prerequisite concepts are learned at least up to the advanced level, weak prerequisite concepts are learned at least up to the surface level
- **Exercise prerequisites:** at the point where an exercise is presented, all strong prerequisite concepts are learned and demonstrated with examples, all weak prerequisite concepts are at either learned or side-demonstrated with examples.

The prerequisite checking on the level of course items is especially important for programming courses that usually have very few direct prerequisite relationships between concepts. Most of the concepts could be introduced independently from other concepts. That's why there could be many possible ways to teach the same subject. However, adopting a particular approach to teaching the subject usually results in lots of indirect prerequisites "hardwired" into educational material. Presentation-level prerequisites are one of the cases of "unnecessary" prerequisites. Another case is example-level or problem-level prerequisites. A concept A does not depend of concept B and could be learned either before or after B. However, in the current course material all available examples or exercises that use B also include A. As a result, the material requires A to be learned *before* B. All these kind of prerequisites are very hard to keep in mind. The only way to ensure that the course is built or redesigned with no prerequisite conflicts is careful prerequisite checking.

Finding content "holes"

A failure to meet the prerequisites could mean either a problem with structure (the item that could meet the prerequisite does exist in the courses but placed after the checked item) or a problem with content (no item to cover the prerequisite). The system can distinguish these two cases and provide a helpful report of a problem. While the former

problem could be often resolved by restructuring the material, the latter indicates a need to expand the course material.

Consolidation of presentations

In a well-designed course each concept has to be presented in full in a single place (subsection or section). It is the place where the student will be returning to refill the gaps in his/her knowledge of a concept. This place is called the concept "host sections". A concept could be introduced before its host section (to enable the student to learn or practice other concepts) but not too many times (hardly more than twice) and not after the full presentation. The system can check these rules using indexing. (Note: The same is not true about examples. It's quite desirable to have several examples for each concept).

Question placement and repositioning

Well-designed quiz questions have one or two outcome concepts. Thus, the system can automatically place new questions into the proper topics by finding the section where the last of these concepts is presented in full. With automatic placement we can delegate course and question design to several authors and get a consistent course. If the course is re-structured the questions can be automatically repositioned.

Guidelines for question design

By matching concepts presented in a topic and concepts assessed by the topic's question pool it is easy to identify a set of concepts that can never be assessed. The identified deficit could drive the question design process. Same procedure can also ensure that the questions in the pool are reasonably evenly distributed among the concepts (to avoid the situation where 80% of questions are testing 20% of concepts).

Matching presentations with examples and exercises

It is possible to check to what extent examples and exercises matches their place in the course and to what extent they cover the presented content. It can be done by matching the set of concepts presented in the section with the joint sets of goal concepts of exercises and examples located in this section. In an ideal situation each section should present, demonstrate (by examples) and assess about the same sets of concepts. If there are too many concepts that are presented but not covered by examples or exercises, the coverage is low. If there are too many concepts that are covered by exercises or examples but not presented in the section (if there is no prerequisite conflict they could be simply presented in previous sections) then the relevance is low. Small mismatch between presentations, examples, and concepts is not a problem, but bigger mismatch in either

direction is a sign of poorly designed section and an indication that something has to be redesigned.

Checking course design against the real course

An author could start the course design with a design document that lists all essential concepts to be introduced in each section. The design document could be stored separately from the course. The system can check how the real course matches the original design by comparing where the author planned to introduce the key concepts and where they are really introduced; how the set of target concepts is supported by questions, examples, and exercises.

Presentation density and sectioning

While different concepts may require different amount of presentation, the overall complexity of a content fragment could be measured by the number of concepts presented in it. By controlling the number of concepts presented in each section we can identify two types of problems: presentation density, where too many concepts are presented in a relatively short section, and uneven distribution of content where number of concepts subsections of the same level

Controlling the difficulty of examples and exercises

Prerequisite indexing of exercises and examples specifies minimal requirements for the concept level that have to be met to make an example or an exercise ready to be taken. Its legal, however that some concepts have higher level of knowledge then it is prescribed. For example, a goal concept of an exercise has to be learned up to the advanced level. In real life, the student reaches this exercise when he or she has already seen several examples with this concept or even solved an exercise involving this concept. It makes this exercise easier for that student. Generally, we can estimate difficulty or learning item by measuring a difficulty between the target state of the goal concepts and the starting state. If all goal concepts or an exercise have been already applied in previous exercises, the exercise is quite simple. If none of them have even be used in examples, the exercise is very difficult. The difficulty of an exercise is not a constant – it depends on the place of the exercise in the course. It makes sense to control the difficulty of examples and exercises in the course to make sure that none example or exercise is too simple or too difficult.

There is research evidence that there exists an optimal difficulty of a learning item for each individual student (i.e., that the student learns best when he or she is presented with learning items with difficulty closed to optimal. We can't use this finding directly since our courses are static – all students go the same way. But in some future we can

found that different groups of users can handle different difficulties. It could be used for making better-targeted courses for special categories of users.

Implementation

The first version of the system was completed in 1999 and evaluated on one of CTE courses. With a help of the system we were able to find and fix a number of problems in the course. The system is written in Java and supports the following functions:

- Prerequisite checking
- Finding content "holes"
- Consolidation of presentations
- Question placement and repositioning

While the system turned out to be very useful, we have encountered a problem. In addition to a good number of real large and small problems the system has also reported a number of problems that no real teacher would count as a problem. It turned out that the course consistency rules behind the system are too rigid. In real life teachers can perfectly tolerate a number of small inconsistencies in the course. Moreover, in some cases the course may be formally "inconsistent" with a purpose. A teacher may want to provoke student thinking by presenting an example that is based on a material that is not yet presented but could be understood by analogy with the learned material. Our quick answer to this problem was color coding the course problem report (Figure 1). In particular, the messages that always report a real problem in the course are colored red not to be missed. The messages that report a problem that often may be tolerable are colored green. We use three to four colors in our reports. A real solution to this problem would be a more precise set of checking rules that is adapted to the course "teaching approach" and, probably, a better indexing.

Prospects

We plan to continue the work on course maintenance system adding features and checking it with an incrementally larger volumes of course material. We see a very important mission in this process. The outcome of this process is not only consistent courses of higher quality, but also a large volume of carefully indexed learning material. Thus we are decreasing bootstrapping cost of more flexible sequencing technologies. We hope that this process will eventually lead to the acceptance of more flexible approaches in large-scale Web-based education.

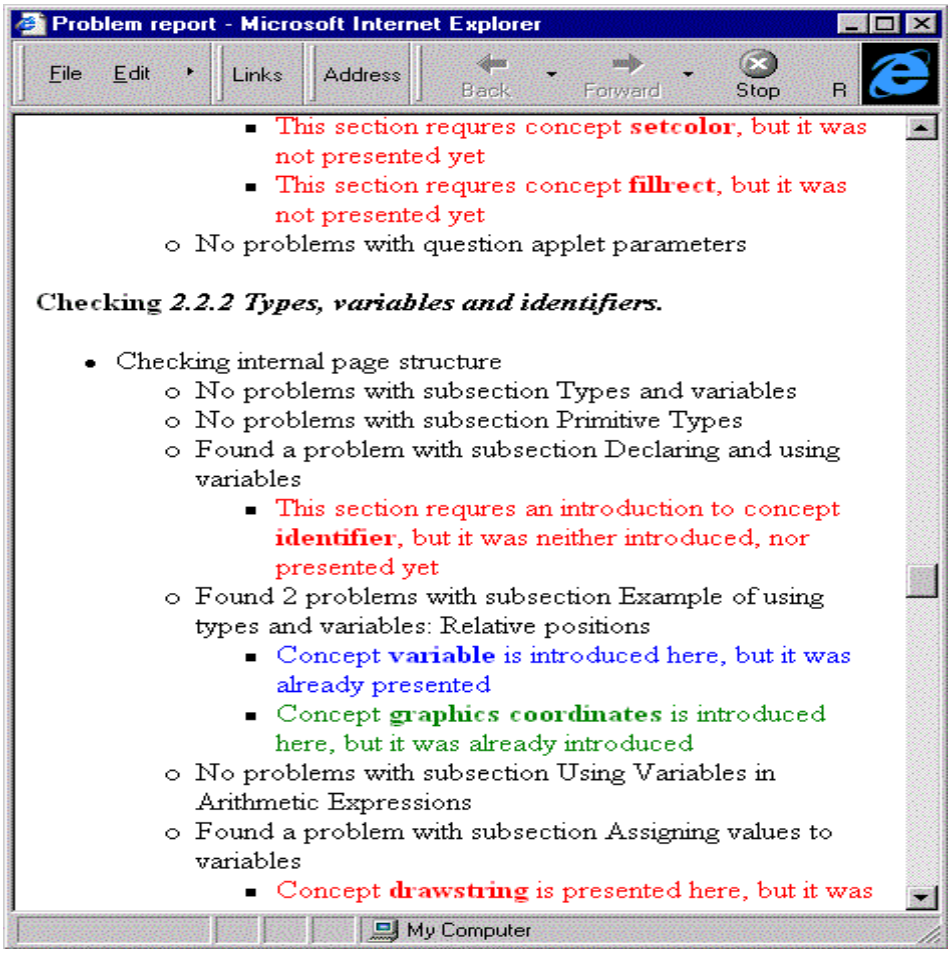

Figure 1. A fragment of a problem report for a Java course

References

[Brusilovsky, Eklund & Schwarz 1998] Brusilovsky, P., Eklund, J., & Schwarz, E. (1998). Web-based education for all: A tool for developing adaptive courseware. Seventh International World Wide Web Conference. Computer Networks and ISDN Systems, 30 (1-7), 291-300.

[Brusilovsky & Miller 2000, in press] Brusilovsky, P., & Miller, P. (2000, in press). Course Delivery Systems for the Virtual University. In Tschang, T., & Della Senta, T. (Eds.), Access to Knowledge: New Information Technologies and the Emergence of the Virtual University. Amsterdam: Elsevier Science.

[Capell & Dannenberg 1993] Capell, P., & Dannenberg, R.B. (1993). Instructional design and intelligent tutoring: Theory and the precision of design. Journal of Artificial Intelligence in Education, 4 (1), 95-121.

[Robson 1999] Robson, R. (1999). WWW-based course-support systems: The first generation. International Journal of Educational Telecommunications, 5 (4), 271-282.