Supporting Adaptive Hypermedia Authors with Automated Content Indexing

Sergey Sosnovsky, Peter Brusilovsky, Michael Yudelson

University of Pittsburgh, School of Information Sciences
135 North Bellefield Avenue, Pittsburgh, PA 15260, USA
{sas15, peterb, mvy3}@pitt.edu

Abstract. The main hindrance to expanded use of adaptive educational hypermedia systems is the need for content to be properly described in the terms of domain concepts. This requirement slows down the authoring process and creates an obstacle to the broader distribution of such systems. In the current paper, we propose an approach to providing automated content indexing for adaptive educational hypermedia systems. Both stages of automated content indexing (content parsing and prerequisite/outcome identification) are described here in detail. The approach we have developed has been implemented by indexing the content of the NavEx system and has proven itself by creating meaningful recourse for adaptive example navigation support.

1 Introduction

More and more adaptive hypermedia systems [2] are reaching the point where they can be used in the context of real education, an area that is now almost exclusively served by traditional non-adaptive web-based educational (WBE) systems [4]. Thanks to years of research, the problems of representing the domain model, knowledge about the student, as well as development of the interface can now be solved in a number of domains by relatively small research teams. The choice of the Web as an implementation platform can help a small team solve problems of delivery, installation, and maintenance, expanding the systems availability to hundreds and thousands of students. Yet, there "the last barrier" exists: The traditional static, non-adaptive WBE systems and courses have something that almost no intelligent system developed by small research teams can offer – large amounts of diverse educational material. A high-quality traditional WBE course may have thousands of presentation pages, and hundreds of other fragments of learning material – examples, explanations, animations, and objective questions created by a team of developers. In comparison, the number of presentation items in the best adaptive web-based educational systems is well under two hundred and the number of other fragments of learning material, such as problems or questions is usually no more than a few dozen. These numbers are certainly sufficient for a serious research study of the system in classroom use, but the number is still small for the needs of practical web-based education, i.e., the ability to support reasonable fragments of practical courses which can be taught to large numbers of students, semester after semester.

We think that the key to solving this last problem is teacher-oriented authoring tools for providing the content for adaptive educational hypermedia systems. The pioneering paper (describing the PAT Algebra tutor [17]) provides a good analysis of problems and a set of design principles developed to use when authoring problems for a cognitive, rule-based tutoring system. The situation described in this paper is when the content to be created is really "intelligent content." The power of intelligent content is that knowledge is hidden behind every fragment of it. Even the simplest "presentation" fragments of external content should be connected to the proper elements of domain knowledge (concepts) so that an intelligent educational system (IES) can understand what it is about, when it is reasonable to present it, and when it is premature. More complicated content format, such as examples and problems, require additional coding in order to enable an IES to run the example or to support the student's ongoing solution of a problem.

There are very few domains where the knowledge behind a fragment of educational content can be deduced automatically by the system from the problem statement. For example, straightforward representations might be used in an IES which teaches derivation in calculus [6], expression evaluation in C [5], or equation-solving in algebra [16]. In these domains it is quite easy to identify rules or concepts behind a problem or an example and no other knowledge except core domain knowledge is required to support the problem solving process. In these "lucky" domains, authoring of additional problems or examples is easy; the author only needs to provide the problem statement in a traditional form. Yet, even in such simple domains, teacher involvement is required for the advanced hypermedia systems to distinguish multiple-concept sequences, such as those including prerequisites or outcomes [3]. In less formalized domains, the knowledge behind a content fragment can't be easily extracted and has to be provided during the authoring process, such as in adaptive hypermedia systems like KBS-Hyperbook [12] or SIGUE [10].

In this paper, we present a collaborative approach to authoring intelligent content for adaptive hypermedia. In our approach the work is distributed between an intelligent authoring system and a teacher. The teacher informs the system about his or her preferred way of teaching by grouping prepared content into a sequence of topics or lecture sets. An intelligent authoring system extracts concepts from the provided fragments of content and classifies them as either prerequisite or outcome concepts on the basis of the teacher's preferred method of teaching. We have applied this approach in our recent system, NavEx.

2. Lack of Authoring Support in the Context of a Programming Course

We have used a number of different web-based educational systems in the context of an undergraduate course "Introduction to Programming" being taught at the University of Pittsburgh's School of Information Sciences. The results of every semester's evaluation and students' feedback led us to assume that the pedagogocal value of at least two of them could be increased by providing a system with

metaknowledge about its content, thus an implementation of adaptive hypermedia technologies. Two of these systems are briefly described below.

The WebEx system serves out interactive, explained examples of programming solutions, via the Web. An author of an example or a later teacher can provide textual explanations for every line of the program code. The students can browse these comments at their own pace and order by selectively clicking on the commented lines (see Fig. 1). The first version of WebEx has been implemented and was reported on in 2001 [9]. Since that time, WebEx has been heavily used. Each semester it has offered an incrementally larger subset of examples from the classroom lectures. The results of evaluation demonstrated that a solid fraction of students wanted to see more explained examples than just the examples from the lecture. Moreover, the proportion of students who wanted "more examples" was growing, even though we were incrementally making more and more lecture examples available each semester. Typically, our course has grown to about 60 examples. This is a relatively large amount, but students have no trouble finding the relevant examples because each example is linked to a specific lecture. In contrast, examples from another class or a digital library are a navigational burden to the student. With no clear guidelines as to which of these examples should be accessed and when, students may easily choose examples that are either too complicated or too simple for the student's current progress.

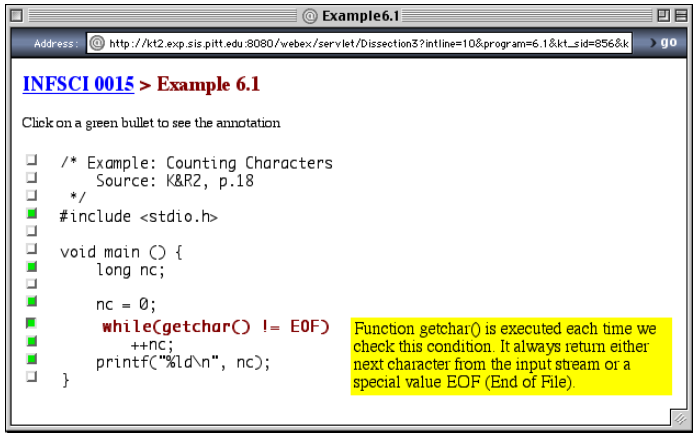

Fig. 1. Interactive example in WebEx.

Another application is QuizPACK. This system authorizes and delivers parameterized web-based quizzes for C-programming courses. The detailed description of QuizPACK can be found in the example for [15]. Figure 2 demonstrates the student interface of the system. QuizPACK evaluation and individual feedback, though positive, showed that students suffer from lack of guidance. Since QuizPACK is used mostly for self-assessment, students need to estimate what their weakest topics are and then decide what quiz is necessary to take, on their own. However, they often get lost "in space" containing about 50 quizzes, which was necessary to develop in order to cover all of the course material. To assist students system needs to provide a valuable feedback which helps them to locate

themselves in the course-knowledge space. Need in adaptive navigation support leads to the necessity of creating "intelligent content."

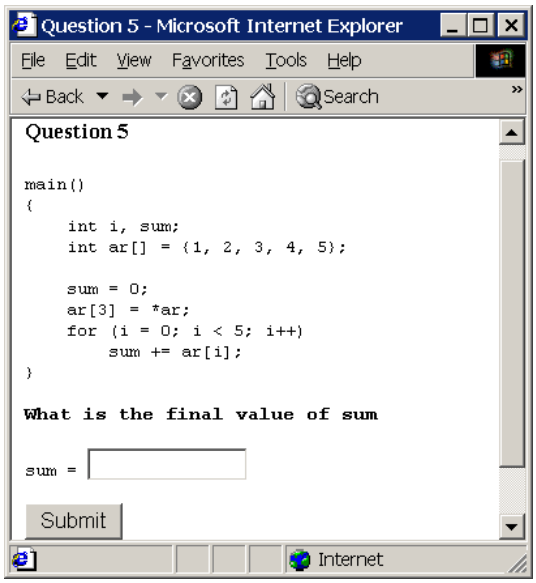

Fig. 2. Typical question in QuizPACK.

3. Automated Content Indexing

There are no universally-accepted recommendations as to which level is best to use when defining a concept in computer programming domains. Some authors suppose that it has to be done on the level of programming patterns or plans [13], other believe, that the concepts is to be closer to the elementary operators [1]. According to the first point of view, the notion of pattern is closer to the real goal of learning programming, since the patterns are what programmers really use. However, the second way is more straightforward and makes the burden of indexing more feasible. With the notable exception of ELM-PE [18], all adaptive sequencing systems known to us work with operator-level concepts. Current implementation of the proposed indexing algorithm also uses the operator-level approach.

This algorithm has two main stages. In the first stage, it extracts concepts from the content elements (examples, questions, presentation pages) combined by the teacher into activity pool. For example, all WebEx examples form one pool; all QuizPACK quizzes belong to another pool. In the second stage, the prerequisite/outcome structure of the course is built in terms of concepts, describing content elements. This sequential structure, along with the indexed content, provides the basis for adaptation. The following two subsections explain both stages in detail.

3.1. On-line Content Parsing

Traditionally, the extraction of grammatically meaningful structures from textual content and the determination of concepts on that basis is a task for the special class of programs called parsers. In our case, we have developed the parsing component with the help of the well-known UNIX utilities: lex and yacc. This component processed source code of a C program and generates a list of concepts used in the program. Currently, fifty-one concepts have been determined for the subset of C language studied during our course. Each programming structure in the parsed content is indexed by one or more concepts, depending upon the amount of knowledge students need to have learned in order to understand the structure. For instance, the parser generates the following list of concepts for the program code in Fig. 1, the WebEx example:

> *include, void, main_func, decl_var, long, decl_var, assign, ne_expr, pre_inc, while, compl_printf*

It is necessary to mention that each concept here represents not simply a keyword, found in the code, but a grammatically complete programming structure. For instance, concept *while* is recognized by the parser only after the whole while-loop, including the keyword *while*, the iteration condition and the loop body, is found. This is why the concept *while* in the index list above must come after the concepts *ne_expr* (not-equal expression: *!=*) and *pre_inc* (pre-increment: *++identifier*).

The parsed code is accessed through the http-protocol. It can be represented as a simple source file in text format or as a properly formatted HTML-file, which distinguishes the code samples from the rest of the HTML content with one of these commonly accepted tag pairs: <code> – </code>, <pre> – </pre>, or <tt> – </tt>. Usage of other HTML-tags, such as the nesting of <b>, <a>, , etc. between the pair of code-tags does not influence the result of the parsing. These tags are simply ignored; at the same time an HTML escape-sequences (like   < & etc.) are processed and converted into corresponding symbols or symbol sequences. For example, Table 1 demonstrates two code samples, which present alternative ways to format HTML code for the QuizPACK question showed in Fig. 2. The parsing component processes all three samples (including the pure C source code of QuizPACK question) in the same way and generates for them the same list of concepts:

> *main_func, int, decl_var, decl_var, int, decl_array, init, assign, assign, derefer, assign, l_expr, post_inc, add_assign, for*

Hence, the web-parser we have developed could be used for indexing the great amount of created on-line C content, such as code libraries and web-based tutorials. Given the URL of the content resource it generates list of concepts, extracted from the C code used in this resource.

Table 1. Two alternative html formats for a simple C program.

<html>
…
<code>
main()
{
  ****int**** i, sum;
  ****int**** ar[] = {1,2,3,4,5};
 	
  sum = 0;
  ar[1] = *ar;
  ****for**** (i = 0; i < 5; i++)
  sum += ar[i];
}
</code>
…
</html>
	<html>
	…
	<pre>
	main()**
	**
	{**
	  **
	int i, sum; **
	  **
	int ar[] = {1,2,3,4,5};
	**
	
	**
	sum = 0; **
	  **
	ar[1] = *ar; **
	  **
	****
	for****(i = 0; i **<** 5; i++)
	**
	    **
	sum += ar[i]; **
	**
	}**
	</pre>
	…
	</html>

3.2. Prerequisite/Outcome Identification

The outcomes of the parsing stage are index lists for all content elements. However, it is not enough for our purposes, since these lists are not yet connected to each other and the content still does not possess any structure on which we could base the adaptation process. In the next stage, all concepts related to each content element are divided into prerequisite and outcome concepts. Prerequisites are the concepts that the student needs to master before starting to work with the current element. Outcomes denote concepts that the element leads one toward learning, i.e. the learning goals of the element.

We use an original algorithm for the automatic identification of outcome concepts (see Fig. 3). This algorithm is flexible enough to be influenced by an instructor-specific way of teaching the course. The source of knowledge for this algorithm is a sequence of groups of content elements. Each group is formed by the elements introduced in the same lecture. Groups are ordered according to the order of lectures in the course, forming a sequential structure of the learning goals of the course. The prerequisite/outcome division algorithm is based on the following assumptions:

- While analyzing content element from some lecture, concepts corresponding to this element and introduced in a preceding lecture are considered to be already learned.
- In each content element, all concepts introduced in the previous lectures are considered as prerequisites, while the concepts first introduced in the current lecture are regarded as outcomes.
- The set of new concepts found in all content elements associated with the lecture becomes the learning goal of the lecture.

The direct outcome of this algorithm is a fully-indexed set of content elements belonging to each lecture and a sequence of learning goals associated with the

lectures. This sequence represents the specific approach to teaching C-programming that is employed by this specific instructor [3]. Once the content elements are indexed and the goal sequence is constructed, any future additional element can be properly indexed by the algorithm and even associated with a specific lecture in the course. More precisely, an association with a specific lecture is the first step in this process. The element is associated with the last lecture that introduces its concepts (i.e., the latest lecture, whose learning goal contains least one concept belonging to this element's index). After that, the element is associated with this lecture. It is important to stress that the outcome identification is adapted to a specific way of teaching a course "mined" from the original sequence of content elements. It is known that different instructors teaching the same programming course may use a very different order for their concept presentation [14]. Naturally, content sequencing in a course is to be adapted to the instructor's way of teaching.

```
learnt_concepts = ∅
for i = 1 to no_of_chapters
{
  for j = 1 to chapter[i].no_of_examples
  {
    chapter[i].example[j].prereq = learnt_concepts ∩ chapter[i].example[j].all_concepts
    chapter[i].example[j].outcome = chapter[i].example[j].all_concepts \ learnt_concepts
  }
  for j = 1 to chapter[i].no_of_examples
    learnt_concepts = learnt_concepts ∪ chapter[i].example[j].all_concepts
}
```

Fig. 3: Pseudo-code for prerequisite/outcome identification.

Although, the described indexing approach, using a parsing component, is specifically for programming and for the learning content based on the programming code (questions, code examples), we believe that the proposed general idea is applicable for a broad class of domains and types of content. In less formalized conditions, where concepts do not have a salient grammatical structure, the classic information retrieval approach could be used instead of parsing.

Currently, the described approach is implemented in the NavEx system, which provides adaptive annotations for programming code examples. The indexed content elements in NavEx are the same programming examples used in WebEx (see Fig. 1). Preliminary evaluation shows that implemented indexing algorithm along with the mechanism for building inter-concept hierarchies from the given, flat content provides meaningful recourse for adaptive example-navigation support in NavEx. The NavEx mechanism of adaptation as well as the system interface will be briefly described in the following section.

4. Adaptive Navigational Support in NavEx

The interactive window of the NavEx system is divided into 3 frames (see Fig. 4). The leftmost frame contains a list of links to all examples/dissections available to a student

in the current course. The links are annotated with colored icons. A red bullet means that the student has not mastered enough prerequisite concepts to view the example. The link annotated with the red bullet is thus disabled. A green bullet means that the student has enough knowledge to view the example. A green check mark denotes that the example has already been seen by the student. A green "play" bullet means that this line of code in the example is currently being viewed. The order of links to examples is fixed, so that students can find them in the same place, no matter what the student's progress through the course has been.

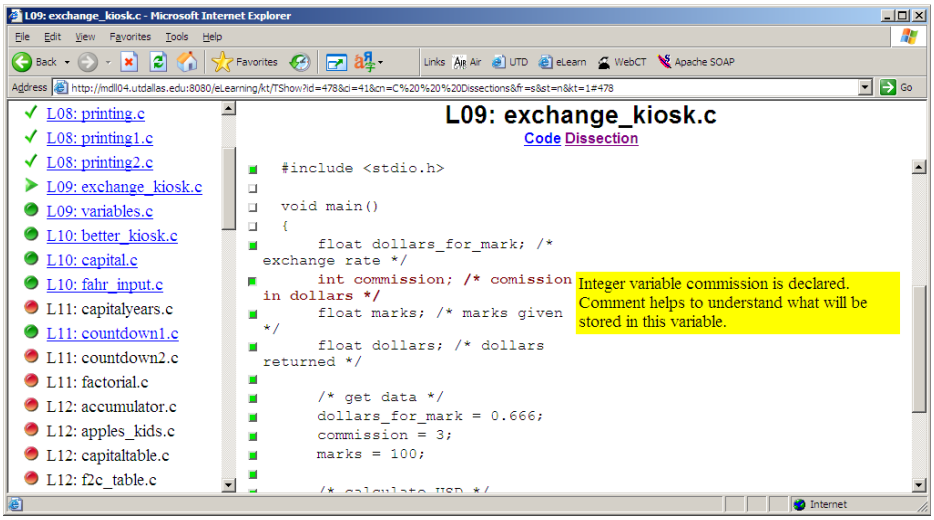

Fig. 4: The interface of NavEx.

The central frame displays the name of the current example. Underneath it are two links: one loads the source code of the example into the central frame (where it will be copied, compiled, and explored); the other link loads interactive example dissection (served directly by the WebEx system). Dissection means that one takes the original source code and comments on it. These comments address the meaning and purpose of this line of code and help the student to understand the example better. Extended comments are shown to the left of the code and can be activated by clicking on the bullet next to the line of the code. If the comment is available, the bullet is green; otherwise, it is white.

NavEx is considered a value-added service of the KnowledgeTree architecture [8], and implements several common protocols including student modeling and transparent authentication. As a typical value-added service, NavEx stands between e-Learning portals and reusable content elements and provides additional value for both teachers and students who use this content through the portal.

Adaptive navigational support in NavEx is done on the basis of the overlay student model [11]. Student's knowledge is represented as a binary vector k, where $k_i = 1$ means that the student has successfully mastered concept i, and $k_i = 0$ means the opposite.

When the student logs into the system a new session is created and information about the current state of his/her knowledge is retrieved from the student model. This information contains concepts the student has mastered and a list of examples the student has reviewed.

Knowing the student's knowledge of each domain concept and the prerequisite-outcome profile of examples, the sequencing mechanism can dynamically compute the current educational status for each example, which is then presented to the student in the leftmost frame of the system window (Fig. 4) in the form of adaptive annotations with the colored icons.

When the student clicks on an example link and reviews the example code, the outcome concepts of the example change their state to "learned" ($k_i = 1$). The changes in the knowledge state of the student are then propagated to the student model. The availability of each new example is determined by checking whether any of the previously unavailable examples now have all of their prerequisite concepts mastered. As the student reviews examples, newer examples become available. The knowledge-based adaptive annotation approach used in NavEx is a variation of a popular adaptive annotation approach introduced originally in the ISIS-Tutor system [7].

5. Summary and Future Work

We have discussed the development of an approach for the automated indexing of content based on C programming code. The first stage of this approach is performed by the implemented parsing component, which is able to extract C concepts from on-line content formatted as HTML or as pure C code. Hence, this tool could be used for indexing the great amount of created on-line C code which is contained in on-line libraries and web-based tutorials.

The second stage is the prerequisite/outcome identification and, building on its base, the inter-concept hierarchical structure of the content. The outcome we receive is the instructor-adaptive structure of the course reflecting his/her way of teaching the course.

The designed approach has been implemented for the NavEx system development. NavEx content being indexed consists of the programming code examples. Preliminary evaluation shows that the implemented indexing algorithm along with the mechanism of building inter-concept hierarchies of the content provide meaningful recourse for adaptive example navigation support in NavEx.

Our next goal is to build-in this algorithm as part of the authoring interface for the next adaptive version of QuizPACK system. Since every QuizPACK question is simply a small C program, we expect our algorithm to work successfully for this application and considerably facilitate the authoring process in QuizPACK.

References

1. Barr A., Beard, M., & Atkinson, R. C.: The computer as tutorial laboratory: the Stanford BIP project. International Journal on the Man-Machine Studies 8, 5 (1976) 567-596.

2. Brusilovsky, P.: Adaptive hypermedia. User Modeling and User Adapted Interaction 11, 1/2 (2001) 87-110, available online at http://www.wkap.nl/oasis.htm/270983
3. Brusilovsky, P.: Developing Adaptive Educational Hypermedia Systems: From Design Models to Authoring Tools. In: Murray, Blessing and Ainsworth (eds.): Authoring Tools for Advanced Technology Learning Environments: Toward cost-effective adaptive, interactive, and intelligent educational software. Ablex, Norwood (2003) 377-409
4. Brusilovsky, P. and Miller, P.: Course Delivery Systems for the Virtual University. In: Tschang, T. and Della Senta, T. (eds.): Access to Knowledge: New Information Technologies and the Emergence of the Virtual University. Elsevier Science, Amsterdam (2001) 167-206.
5. Brusilovsky, P. and Su, H.-D.: Adaptive Visualization Component of a Distributed Web-based Adaptive Educational System. In: Intelligent Tutoring Systems. Vol. 2363. Springer-Verlag, Berlin (2002) 229-238
6. Brusilovsky, V.: Task sequencing in an intelligent learning environment for calculus. In: Proc. of Seventh International PEG Conference, Edinburgh (1993) 57-62
7. Brusilovsky, P. & Pesin, L.: Adaptive navigation support in educational hypermedia: An evaluation of the ISIS-Tutor. Journal of Computing and Information Technology 6, 1 (1998) 27-38.
8. Brusilovsky, P. & Nijhawan, H.: A Framework for Adaptive E-Learning Based on Distributed Re-usable Learning Activities. In: Driscoll, M. and Reeves, T. C. (eds.) Proc. of E-Learn 2002, Montreal, Canada, AACE (2002) 154-161.
9. Brusilovsky, P.: WebEx: Learning from examples in a programming course. In: Fowler, W. and Hasebrook, J. (eds.) Proc. of WebNet'2001, World Conference of the WWW and Internet, Orlando, FL, AACE (2001) 124-129.
10. Carmona, C., Bueno, D., Guzman, E., and Conejo, R.: SIGUE: Making Web Courses Adaptive. In: De Bra, P., Brusilovsky, P. and Conejo, R. (eds.) Proc. of Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'2002) Proceedings, Málaga, Spain (2002) 376-379
11. Greer, J. & McCalla, G. (eds.): Student modelling: the key to individualized knowledge-based instruction. NATO ASI Series F, Vol. 125, Springer-Verlag, Berlin (1993).
12. Henze, N. and Nejdl, W.: Adaptation in open corpus hypermedia. International Journal of Artificial Intelligence in Education 12, 4 (2001) 325-350, available online at http://cbl.leeds.ac.uk/ijaied/abstracts/Vol_12/henze.html
13. Lutz, R.: Plan diagrams as the basis for understanding and debugging pascal programs. In: Eisenstadt, M., Keane, M. T. and Rajan, T. (eds.): Novice programming environments. Explorations in Human-Computer Interaction and Artificial Intelligence. Lawrence Erlbaum Associates, Hove (1992) 243-285.
14. Moffatt, D. V. & Moffatt, P. B.: Eighteen pascal texts: An objective comparison. ACM SIGCSE bulletin 14, 2 (1982) 2-10.
15. Pathak, S., Brusilovsky, P.: Assessing Student Programming Knowledge with Web-based Dynamic Parameterized Quizzes. In Proceedings of ED-MEDIA'2002, Denver, Colorado, USA: AACE, 1548-1553.
16. Ritter, S. and Anderson, J. R.: Calculation and strategy in the equation solving tutor. In: Moore, J. D. and Lehman, J. F. (eds.) Proc. of the Seventeenth Annual Conference of the Cognitive Science Society, Hillsdale, NJ, Erlbaum (1995) 413-418
17. Ritter, S., Anderson, J., Cytrynowicz, M., and Medvedeva, O.: Authoring Content in the PAT Algebra Tutor. Journal of Interactive Media in Education 98, 9 (1998), available online at http://www-jime.open.ac.uk/98/9/
18. Weber, G. & Bögelsack, A.: Representation of programming episodes in the ELM model. In: Wender, K. F., Schmalhofer, F. and Böcker, H.-D. (eds.): Cognition and Computer Programming. Ablex, Norwood, NJ (1995) 1-26.