An Adaptive E-Learning Service for Accessing Interactive Examples

Peter Brusilovsky, Sergey Sosnovsky, Michael Yudelson
University of Pittsburgh, USA
{peterb, sas15, mvy3}@pitt.edu

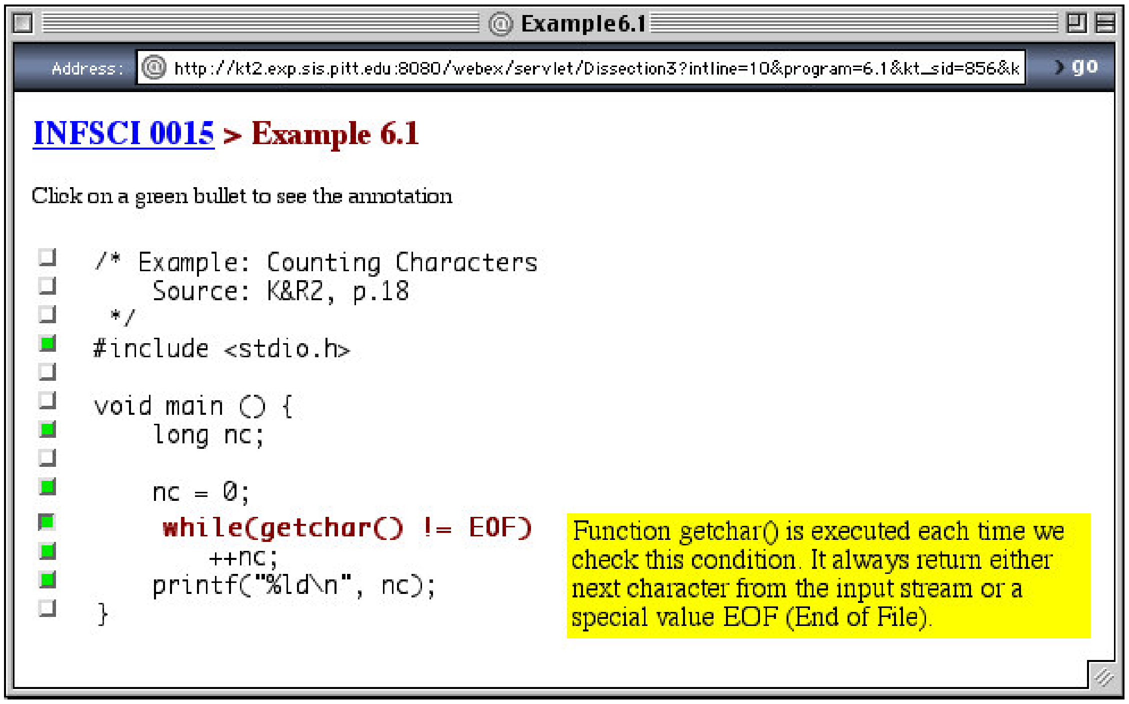
Abstract: This paper discusses the need to provide adaptive navigational support for students accessing large numbers of interactive examples in Web-enhanced education. Here we introduce the system NavEx (Navigation to Examples), which is able to provide the adaptive annotation of programming examples without the need for manual indexing. NavEx uses innovative algorithms for the prerequisite-outcome indexing of learning materials and is an efficient knowledge-based approach for adaptive annotation. The system serves as an adaptive value-added service residing between an learning portal and an advanced re-usable content.

1. Web-based Examples in a Programming Course

It has often been claimed that humans use solutions to previous problems to solve new problems or planning tasks. This is especially pertinent to the domain of programming. Both experienced and novice programmers often use program examples they have created or learned in the past to solve new programming tasks. Experienced teachers of programming courses present and explain one or more carefully chosen example programs during every lecture. In Web-based and Web-enhanced education the code of these examples is provided on the Web, along with other kinds of learning material, in order for the students to study and run the example code on their own. Our system, WebEx (Brusilovsky 2001), attempts to integrate benefits of these two approaches (instructor explanation and web interactivity) into example presentation. WebEx is able to serve interactive, explained examples via the Web. The idea of WebEx is simple: the author or later the instructor using the example code can provide textual explanations for every line of the program code. The students can browse these comments at their own pace and sequence, by selectively clicking on the commented lines (Figure 1).

The first version of WebEx was implemented and reported in 2001 (Brusilovsky 2001). Since then, WebEx has been heavily used every semester in undergraduate programming courses at the University of Pittsburgh. Each semester, WebEx has offered an incrementally larger subset of examples from the classroom lectures in the commented, interactive format. We have run several questionnaire-based formative evaluations of WebEx over the last semesters. In their free-form answers to the first two questionnaires a few students expressed interest in seeing more commented programming examples, rather than just the 3-5 examples offered for each lecture. To evaluate the need for increasing the number of examples, we have inserted a new question into most recent questionnaires. We asked the students what kind of examples they would like to see presented through WebEx. The answers ranged from “No examples at all” to “All examples from the lecture” to “Additional examples beyond the ones provided at the lecture.” The results indicated 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 want "more examples" has been growing even while we were increasing the number of lecture examples available each semester. For example, in the Fall 2002 semester, only 39% of respondents were interested in examples “beyond lectures” while 61% were interested in having lecture examples only. In the Spring 2003 semester, however, the number increased: 50% voted for “more examples” and 50% wanted nothing but lecture examples (in both semesters no one selected the “No examples” option.

Since WebEx is a multi-author system that is used by several instructors, providing “more examples” may look like a trivial problem. In a typical situation when different instructors use their own examples, the only thing they need to do in order to offer more explained examples to their classes is to agree about how to use examples from other instructors. One may even add that, in the future, a faculty using WebEx should be able to get examples prepared by other faculty from an educational digital library (such as http://www.citidel.org/). However, the lack of navigation support for the students makes this “simple” example-sharing solution unusable.


Example6.1
Address: http://kt2.exp.sis.pitt.edu:8080/webex/servlet/Dissection3?intline=10&program=6.1&kt_sid=856&k
go

INFSCI 0015 > Example 6.1

Click on a green bullet to see the annotation

```
/* Example: Counting Characters
    Source: K&R2, p.18
 */
#include <stdio.h>

void main () {
    long nc;

    nc = 0;
     while(getchar() != EOF)
        ++nc;
    printf("%ld\n", nc);
}
```

Function getchar() is executed each time we check this condition. It always return either next character from the input stream or a special value EOF (End of File).

Figure 1: The interface of WebEx

Our typical course has about 60 examples. This is a relatively large amount, but students have no trouble finding relevant examples because each example is linked to a specific lecture. Working on the topics of each new lecture, the student can focus on 2-5 examples associated with this lecture by the teacher. In contrast, examples from another class or a digital library are a navigational burden. Given no clear guidelines as to which of these examples to access and when, students will most likely choose examples that are inappropriate at the given moment (such as being too complicated, too simple, or the wrong subject matter).

To help the student to select the "right" example, we applied adaptive, prerequisite-based annotation, a specific adaptive navigation support technology (Brusilovsky 1996). This technology was first explored in ISIS-Tutor system (Brusilovsky et al. 1998b) and was found very efficient. Since then, it has been used in ELM-ART (Weber et al. 2001), InterBook (Brusilovsky et al. 1998a), ACE (Specht et al. 1998), and other systems. With this technology, each learning object is described in terms of domain concepts. Some concepts serve as learning goals and others as prerequisites to understanding the object. Tracking the student’s current level of knowledge, an adaptive system can dynamically classify examples as being relevant or not (e.g., too simple or too advanced). The status of each object is shown by using a traffic light metaphor.

This paper presents our system NavEx (Navigation to Examples) which was designed to explore the idea of providing adaptive navigation support for accessing programming examples. The next section presents the NavEx system from the student’s point of view. The rest of the paper introduces the technologies that we developed to implement NavEx.

2. The NavEx Interface

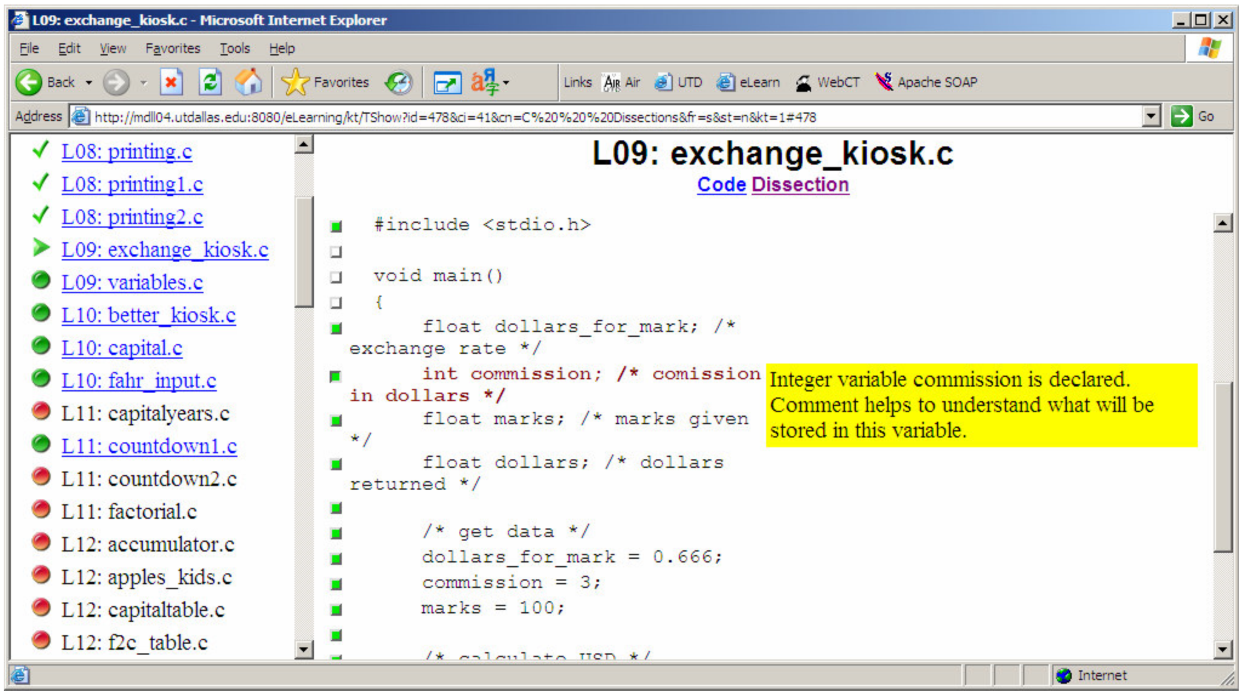
The interactive window of the NavEx system is divided into 3 frames (Figure 2). The leftmost frame contains a list of links to all the examples/dissections available for a student in the current course. The links are annotated with colored bullets. A red bullet means that the student has not mastered enough prerequisite concepts to view the example. Therefore, the link annotated with the red bullet is disabled. A green bullet means that the student has enough knowledge to view the example. The green check mark shows that the example has already been seen by the student. A green “play” bullet shows that the example is currently being viewed. The order of example links is fixed, so that students can more easily find them in the same place, despite the student’s progress through the course.

The upper frame displays the name of the current example. Underneath it are two links: one loads the source code of the example to the central frame (to be copied, compiled, and explored), the other one loads an interactive example *dissection* (served directly by the WebEx system, which is now a component of NavEx). Dissection includes the same source code but now it has line comments from the author or a later instructor who is using the example. These comments address the meaning and purpose of that line of code and help to explain the current concept being taught. Extended comments are shown to the right of the code and can be activated by clicking on the bullet next to the line of code. If a comment is available, the bullet is green; if not, the bullet is white (symbolizing emptiness).

NavEx interface is implemented as a server-side solution written in Java. All knowledge and data are stored in a relational database. NavEx is considered to be a value-added service of the KnowledgeTree architecture (Brusilovsky et al. 2002), and implements several common protocols, including student modeling and transparent authentication. As a typical value-added service, NavEx resides between E-Learning portal and re-usable content objects and provide additional value for teachers and students who use this content through the portal. Unlike other kinds of value-added services, such as annotation services, the value added by NavEx is the ability to adapt to the course goals and student knowledge. With NavEx, teachers can bypass the time-consuming process of selecting examples for each course lecture that meet goal and prerequisite restrictions. Students receive adaptive guidance in selecting examples that are most relevant to their learning goals and knowledge.

3. Indexing Examples in NavEx

To provide an adaptive guidance for the student, a system needs to have knowledge about the material itself. Indexing learning material with domain knowledge is a time-consuming process requiring expertise in both the domain of learning and knowledge engineering (Brusilovsky 2003). In NavEx we explored an alternative automated approach. The current section describes this process in detail.


L09: exchange_kiosk.c - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Favorites Links Air UTD eLearn WebCT Apache SOAP
Address http://mdll04.utdallas.edu:8080/eLearning/kt/TShow?id=478&ci=41&cn=C%20%20%20Dissections&fr=s&st=n&kt=1#478 Go
Internet

✓ L08: printing.c
✓ L08: printing1.c
✓ L08: printing2.c
➤ L09: exchange_kiosk.c
● L09: variables.c
● L10: better_kiosk.c
● L10: capital.c
● L10: fahr_input.c
● L11: capitalyears.c
● L11: countdown1.c
● L11: countdown2.c
● L11: factorial.c
● L12: accumulator.c
● L12: apples_kids.c
● L12: capitaltable.c
● L12: f2c_table.c

L09: exchange_kiosk.c

Code Dissection

```
  #include <stdio.h>

  void main()
  {
      float dollars_for_mark; /*
exchange rate */
      int commission; /* comission
in dollars */
      float marks; /* marks given
*/
      float dollars; /* dollars
returned */

      /* get data */
      dollars_for_mark = 0.666;
      commission = 3;
      marks = 100;

```

Integer variable commission is declared. Comment helps to understand what will be stored in this variable.

Figure 2: The Interface of NavEx.

There are no universally accepted recommendations about which level is better for defining concepts in programming domains. Some authors believe that it has to be done on the level of programming patterns or plans (Lutz et al. 1992); other believe, that concepts should be derived from language constructs (Bar et al. 1976). From the first point of view, the notion of pattern is more fundamental to the real goal of learning programming and 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 (Weber et al.1995), all adaptive sequencing systems known to us work with construct-level concepts. The current implementation of the indexing algorithm for NavEx also uses the construct-level approach.

Traditionally, the extraction of a grammatically meaningful structure from programming code is the task for the special class of programs called parsers. For the NavEx system we have developed an indexing component, which parses the C code of each example and generates the list of used concepts. This component is developed with the help of the well known Unix utilities, lex and yacc.

Fifty-one concepts have been determined to encompass the subset of C language learned during an introductory programming course. Each programming structure in an example can be indexed by one or more concepts depending upon the amount of knowledge the student needs to understand it. As an example, for the program code presented in Figure 1, the following ordered list of concepts is generated:
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, the concept *while* is recognized by the indexing algorithm only after the whole loop structure, including the keyword *while*, the iteration condition and the loop body, is found. That is why the concept *while* in the index list above follows the concepts ne_expr (not-equal expression) and *pre_inc* (pre-increment*)*.

The next stage is to divide the concepts related to each example into prerequisite and outcome concepts. Prerequisites are the concepts that the student needs to master before beginning to work with this particular example. Outcomes denote concepts that the example helps to learn, i.e. the pedagogical goals of the example. As an automatic knowledge extractor, our grammar-based extractor is unable to distinguish prerequisites and outcomes - it simply produces the list of all concepts. For adaptive sequencing and navigation support, however, clear separation of prerequisites and outcomes is vital.

NavEx uses an original algorithm for automatic identification of outcome concepts (Figure 3). This algorithm adapts to the instructor's specific method of teaching the course. The source of knowledge for this algorithm is a sequence of example groups. Each group is formed by examples introduced in the same lecture. Groups are ordered according to the order of lectures in the course. The design of our prerequisites/outcomes division algorithm is based on the following assumptions:

- While analyzing examples from some lecture, concepts corresponding to examples from all preceding lectures are considered to be completely learned.
- In each example, all concepts introduced in the previous lectures are considered to be prerequisites to this concept, while the concepts first introduced in the current lecture as viewed as outcomes.
- The set of new concepts found in all examples associated with the lecture is considered to be the pedagogical goal of the lecture

The direct outcomes of this algorithm are a fully-indexed set of lecture examples and a sequence of learning goals associated with the course lectures. This automatically-generated sequence represents the instructor's specific approach to teaching C programming (Brusilovsky 2003). Once the course examples are indexed and the goal sequence is constructed, any programming example can be properly indexed by the algorithm and even associated with a specific lecture in the course. More precisely, creating an association to a specific lecture is the first step in this process. The example is associated with the last lecture that introduces the example concepts (i.e., the most advanced lecture that has at least one example concept in its pedagogical goal). After that, the example is indexed as belonging to this lecture. It is important to stress again that outcome identification is adapted to a specific way of teaching a course, which is "mined" from the original sequence of examples. It has been known for a long time that different instructors presenting the same programming language may use very different orders of concept presentation (Moffatt et al. 1982). Naturally, example sequencing in a course has to be adapted to the instructor's method of teaching.

```
$learnt\_concepts = \varnothing$
$for\ i = 1\ to\ no\_of\_chapters$
{
  $for\ j = 1\ to\ chapter[i].no\_of\_examples$
  {
    $chapter[i].example[j].prereq = learnt\_concepts \cap chapter[i].example[j].all\_concepts$
    $chapter[i].example[j].outcome = chapter[i].example[j].all\_concepts \setminus learnt\_concepts$
  }
  $for\ j = 1\ to\ chapter[i].no\_of\_examples$
    $learnt\_concepts = learnt\_concepts \cup chapter[i].example[j].all\_concepts$
}
```

Figure 3: Pseudocode for prerequisites/outcomes identification.

Although, the described indexing approach, using the parsing component, is specific for programming, we believe that the proposed general idea is applicable for a broad class of domains. In less formalized domains, where concepts do not have a salient grammatical structure, a classic information-retrieval approach could be used instead of parsing. Current evaluation shows that implemented indexing algorithms, along with the NavEx mechanism of building inter-concept-example hierarchies, provide meaningful recourse for adaptive example navigation support. The mechanism of adaptation is explained in the next section.

4. Providing Adaptive Navigation Support for Examples

Adaptation of navigation in NavEx is done on the basis of the overlay user model (Greer et al. 1993). User knowledge is represented as a binary vector k, where k_i=1 means that the user has successfully mastered concept i, and k_i=0 means the opposite.

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

Knowing the user-knowledge of each domain concept and the prerequisite-outcome profile of all examples, the sequencing mechanism can dynamically compute the current educational status for each example. The current status of each example is presented to the user in the leftmost frame of the system window (Figure 2) in the form of adaptive annotations. Already mastered examples are annotated with green checkmarks, available examples are annotated with green bullets, unavailable ones with red bullets.

When the user clicks on an example link and reviews the resulting example code, the outcome concepts of the example change their state to “learned” (k_i=1). The changes in the knowledge state of the user are then propagated to the user records. The availability of new examples is determined by checking whether any of the previously unavailable examples now have all of their prerequisite concepts mastered. As the user reviews examples, more new 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. This approach is known to be very efficient (Brusilovsky et al. 1998b).

5. Summary and Future Work

We have presented the NavEx system, our first attempt to provide adaptive navigation support for accessing programming examples. NavEx uses an efficient adaptive, prerequisite-based annotation approach. However, unlike other systems that use this approach, NavEx does not require any manual indexing of educational objects. As a result, NavEx can be used by virtually any teacher of a programming course. NavEx allows teachers unfamiliar with adaptive hypermedia to add their own examples, exchange their examples, or use examples from a digital library.

Our current goal is to improve the quality of adaptive navigation support in NavEx. The current version helps to distinguish used, ready, and not-ready examples. The next version will recognize examples that are too simple for the student and also recommend examples that are relevant to the current learning goal. We also plan to connect example exploration with the self-assessment of knowledge using our system QuizPACK (Sosnovsky et al. 2003).

6. References

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

Brusilovsky, P. (1996): Methods and techniques of adaptive hypermedia. User Modeling and User-Adapted Interaction **6**, 2-3 (1996) 87-129.

Brusilovsky, P. (2001): 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.

Brusilovsky, P. (2003): Developing Adaptive Educational Hypermedia Systems: From Design Models to Authoring Tools. In: Murray, T., Blessing, S. and Ainsworth, S. (eds.): Authoring Tools for Advanced Technology Learning Environments: Toward cost-effective adaptive, interactive, and intelligent educational software. Ablex, Norwood (2003) In Press.

Brusilovsky, P., Eklund, J., and Schwarz, E. (1998): Web-based education for all: A tool for developing adaptive courseware. Computer Networks and ISDN Systems. **30**, 1-7 (1998) 291-300

Brusilovsky, P. & Nijhawan, H. (2002): A Framework for Adaptive E-Learning Based on Distributed Re-usable Learning Activities. In: Driscoll, M. and Reeves, T. C. (eds.) Proc. of World Conference on E-Learning, E-Learn 2002, Montreal, Canada, AACE (2002) 154-161.

Brusilovsky, P. & Pesin, L. (1998): Adaptive navigation support in educational hypermedia: An evaluation of the ISIS-Tutor. Journal of Computing and Information Technology **6**, 1 (1998) 27-38.

Greer, J. & McCalla, G. (eds.) (1993): *Student modelling: the key to individualized knowledge-based instruction*. NATO ASI Series F, Vol. 125, Springer-Verlag, Berlin (1993) p.

Lutz, R. (1992): 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.

Moffatt, D. V. & Moffatt, P. B. (1982): Eighteen pascal texts: An objective comparison. ACM SIGCSE bulletin **14**, 2 (1982) 2-10.

Sosnovsky, S., Shcherbinina, O., and Brusilovsky, P. (2003): Web-based parameterized questions as a tool for learning. In: Rossett, A. (ed.) Proc. of World Conference on E-Learning, E-Learn 2003, Phoenix, AZ, USA, AACE (2003) 309-316.

Specht, M. & Oppermann, R. (1998): ACE - Adaptive Courseware Environment. The New Review of Hypermedia and Multimedia **4** (1998) 141-161.

Weber, G. & Bögelsack, A. (1995): 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.

Weber, G. & Brusilovsky, P. (2001): ELM-ART: An adaptive versatile system for Web-based instruction. International Journal of Artificial Intelligence in Education **12**, 4 (2001) 351-384, available online at http://cbl.leeds.ac.uk/ijaied/abstracts/Vol_12/weber.html.