

Interactive Authoring Support for Adaptive Educational Systems

Peter Brusilovsky, Sergey Sosnovsky, Michael Yudelson, Girish Chavan
School of Information Sciences, University of Pittsburgh
135 North Bellefield Ave. Pittsburgh, PA 15260, USA
{peterb, sas15, mvy3} @pitt.edu, chavang@upmc.edu

Abstract. A well-known challenge of adaptive educational systems is the need to develop intelligent content, which is very time and expertise consuming. In traditional approaches a teacher is kept at a distance from intelligent authoring. This paper advocates the involvement of teachers in creating *intelligent content*. We are presenting an approach to the development of intelligent content as well as an authoring tool for teachers that support our approach. This approach has two main stages: elicitation of concepts from content elements and the identification of a prerequisite/outcome structure for the course. The resulting sequence of adaptive activities reflects the author's view of the course's organization. The developed tool facilitates concept elicitation in two ways: it provides an author with an automatic indexing component and also allows her/him to edit the index using the domain ontology as an authoring map.

Introduction

An increasing number of adaptive and intelligent web-based educational systems [1] are reaching the point where they can be used in the context of a real classroom or online school, an area that up to now has been almost exclusively served by traditional non-intelligent and non-adaptive web-based educational systems [2]. Thanks to years of research, a multiple set of problems: representing the domain model, the procedural expertise, the knowledge about a student, as well as developing the interface can now be solved in a number of domains by relatively small research teams. The choice of the Web as implementation platform can help a small team solve problems of delivery, installation, and maintenance, thus making their intelligent systems available to hundreds and thousands of students. Yet, there is one "last barrier." Traditional static, non-intelligent web-based educational (WBE) systems and courses have provided something that almost no intelligent system developed by a small research team 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, including examples, explanations, animations, and objective questions created by a team of developers. In comparison, the number of presentation items in even the best intelligent WBE systems is well under one hundred and the number of other fragments of learning material, such as problems or questions, is no more than a few dozen. These numbers are certainly sufficient for a serious classroom study of the system, but still quite far from the resources needed for a practical web-based educational approach, namely one, which could support reasonable fragments of practical courses that are taught to large numbers of students, semester by semester.

The origin of this bottleneck is the established design paradigm of existing adaptive and intelligent educational systems. With this approach, a system is created by a team of expert developers and shipped to their users (teachers and students) as a whole. Within this approach, little can be done to magnify the volume of available educational content. We think that the move of adaptive and intelligent web-based educational systems (AIWBES) from labs to

regular classrooms has to be supported by a change in the design paradigm. A significant increase of the role of teachers as principal users of intelligent educational systems must be supported by a parallel increase in their participation in the authoring process. We argue that the new paradigm would make teachers more active players in the authoring process by separating the authoring process into two parts: core AIWBES authoring and educational content authoring. Core authoring should comprise the development of the core functionality of AIWBES: knowledge representation, algorithms, interfaces, and core educational content. This part is not different from traditional authoring and should remain in the hands of a professional development team (which hopefully will include some prize-winning teachers). At the same time, the core of AIWBES should be designed in such a way as to allow the majority of the educational content (such as explanations, examples, problems) to be authored by teachers working independently of the development team (and possibly continuing long after the system is originally deployed).

The idea of involving teachers as content authors comes naturally to the developers of the practical AIWBES that are used in dozens of classrooms. It is not surprising that the first implementation of this idea by Ritter et al. [3] was done in the context of the PACT Algebra Tutor, the first AIWBES to make a leap from the lab to hundreds of classrooms [4]. Later, this idea was also explored in the context of AnimalWatch [5], another practical algebra tutoring system. This solution looks like it may be a silver bullet. Not only does it solve the “lack of content” bottleneck, but it also offers multiple additional benefits. The ability to contribute their favorite content transforms teachers from passive users of new technology into active co-authors. It turns an AIWBES which competes with the teacher into a powerful tool in the teacher’s hands. A strong feature of traditional non-adaptive web-based educational systems is that while offering a core framework for web-based education, they also allow every teacher to author easily their own educational content. An AIWBES that allows teachers to add their own content will have a much better chance to compete with the non-intelligent systems which now dominate the educational arena.

The goal of this project is to investigate the use of teachers to develop educational content in a specific domain for AIWBES. The next section discusses the problems faced when supporting teachers as authors of intelligent content. The following sections explain how we address some of these challenges in an authoring system that creates advanced content in AIWBES for an introductory programming class. At the end, we summarize our results and discuss future work.

1. Supporting teachers as authors of adaptive and intelligent content

The teacher's involvement in the process of AIWBES authoring is recognized as both a need and as a research stream in AIWBES community. However, the original goal was also to involve teachers in the *core design* process. This direction of work brought little practical success. After a number of attempts to turn teachers into key developers of AIWBES, no one has the illusion that a teacher can design an AIWBES, even with the help of advanced authoring tools. As pointed out by Murray in his comprehensive overview of ITS authoring tools [6]: "The average teacher should not be expected to design ITSs any more than the average teacher should be expected to author a textbook in their field".

The new design paradigm offers teachers a different place in the process of AIWBES authoring. It leaves the core authoring in the hands of well-prepared design teams and gives teachers a chance to extend the system and fine tune it to their local needs by adjusting and adding to the educational content. Such division of labor is quite natural. Indeed, while it is rare for teachers to be able to create a textbook for their courses, many of them augment

existing textbooks with their own examples, problems, questions, and even additional explanations of complicated concepts.

Still, the development of the content authoring tools for an AIWBES that can be used by regular teachers is a research problem that should not be underestimated. Teachers are much less prepared to handle the authoring than professional AIWBES developers, and require a significant level of support. The pioneering paper [3] provides a good analysis of problems and a set of design principles developed for solving the authoring problems that exist for a cognitive rule-based tutoring system.

The main issue here is that the content to be created for an AIWBES is really *intelligent content*. The power of intelligent content is in the knowledge behind its every fragment. Even the simplest presentation fragments of external content should be connected to the proper elements of domain knowledge (concepts) so that an AIWBES can understand what it is about, when it is reasonable to present it, and when it is premature. More complicated types of content, such as examples and problems, require that even more knowledge be represented, in order to enable an AIWBES to run the example or to support the student while he/she is solving a problem.

For example, adaptive educational hypermedia systems such as InterBook [7], AHA! [8], or KBS-Hyperbook [9] require every hypermedia page to be connected to a domain model concept in order for the server to know when to present them in an adaptive manner. Moreover, InterBook and AHA! require separating connected concepts from page prerequisites (concepts to know before reading a page) and page outcomes (concepts presented in the page). This knowledge has to be provided during the authoring process. As we have found during our work with InterBook, content authors have problems identifying concepts associated with content pages even if the number of concepts in the domain model is under 50. For adaptive hypermedia authoring this “concept indexing” becomes a major bottleneck. While a few pioneer systems such as KBS-Hyperbook [9] and SIGUE [10] allowed teachers to add additional content by indexing content pages with domain concepts, they provide no special support for teachers in the process of indexing. The AHA! System shows some progress towards this goal by providing a graphical authoring tool that will show connections between concepts and pages, but this tool becomes difficult to use when the number of concepts and pages approaches the level of that used in a practical classroom.

Traditionally, there are two ways to support humans in performing complicated tasks: an AI approach (i.e., make an intelligent system that will do this task for the user) and an HCI approach (i.e., provide a better interface for the humans to accomplish the task). In the case of indexing, it means that one must either develop an intelligent system that can extract concepts from a fragment of content or develop a powerful interface that can help the teacher do this manually. While both approaches are feasible, our team was most interested in a hybrid approach – a “cooperative” intelligent authoring system for the teachers that split the work between a human author and an intelligent tool so that both “agents” were able to “cooperate.” doing their share of work. We have started to explore this idea by developing a cooperative authoring system for the domain of programming. The goal of this system is to allow authors to collaboratively index interactive educational content (such as program examples or exercises) with domain model concepts while separating them into prerequisite and outcome concepts.

The following two sections describe our indexing approach and the system that implements it. These sections present two main stages of the approach: concept elicitation and prerequisite/outcome identification. In the first stage, a cooperative indexing tool extracts concepts from the content elements (examples, questions, presentation pages), grouped by the type of activity (i.e., all examples form one pool while all quizzes belong to another pool). In the second stage, a teacher-driven prerequisite/outcome identification algorithm separates the

concepts connected with each content item into prerequisites and outcomes as required by the adaptive hypermedia system. While the cooperative authoring process has been used with two kinds of educational content, the following sections focus on one of these kinds – parameterized quizzes served by the QuizPACK system [11].

2. Content Indexing

There are no universally accepted recommendations as to which level is best to use when defining a concept in the computer programming domains. Some authors theorize that it has to be done on the level of programming patterns or plans [12]. Others believe that the main concepts should be related to the elementary operators [13]. According to the first point of view, the notion of pattern is closer to the real goal of studying programming, since 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 [14], all adaptive sequencing systems known to us work with operator-level concepts. Our web-based cooperative indexing system allows us to combine two kinds of indexing. Simple operator-level indexing is performed by an automatic concept extractor, while more complicated higher-level indexing is performed by the author, using a graphical ontology-based tool.

Figure 1 demonstrates the interface for authoring QuizPACK parameterized questions. The main window is divided into two parts. The left part contains functionality for editing the text and different parameters of the question (details are not important for the topic of this paper). The right part facilitates the elicitation of the concepts used in the question. It provides an author with non-exclusive possibilities: to extract concepts automatically and/or to use a visual indexing interface based on the visualized ontology of available concepts. The following subsections discuss both modes.

2.1. Automated Concept Extraction

Traditionally, the automatic 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 two well-known UNIX utilities: *lex* and *yacc*. This component processes the source code of a C program and generates a list of concepts used in the program. Currently, about 80 concepts can be identified by the parser. Each language 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 list of concepts in the right part of Figure 1 has been generated by the parser for the program code of the question in the left part of the figure. It is necessary to mention that each concept in this list represents not simply a keyword, found in the code, but a grammatically complete programming structure.

To launch the automatic indexing, an author clicks on the button *Extract* under the *Concepts* section of the interface. The list is then populated and the button dims out. If the code of a question has been changed, the button regains its clickability. This is done to prevent the author from losing the results of manual indexing, described in the next subsection.

2.2. Ontology as a Tool for Authoring Support

Automated indexing is not always feasible. Some higher order concepts involve understanding programming semantics that might be hard to extract. In more advanced courses like *Data Structure* or *Algorithm Design*, pattern-oriented questions may be popular. For example, there are several modifications of the sentinel loop. The parser we developed

easily breaks such fragments of code into syntax concepts (which must be learned in order to understand the code), however, it is not reasonable to make it follow each and every possible configuration of the sentinel loop. We also should take into account that an author of content might not fully agree with the results of indexing. She may assume some extracted concepts to be irrelevant, or unimportant, or might want to add some other concepts.

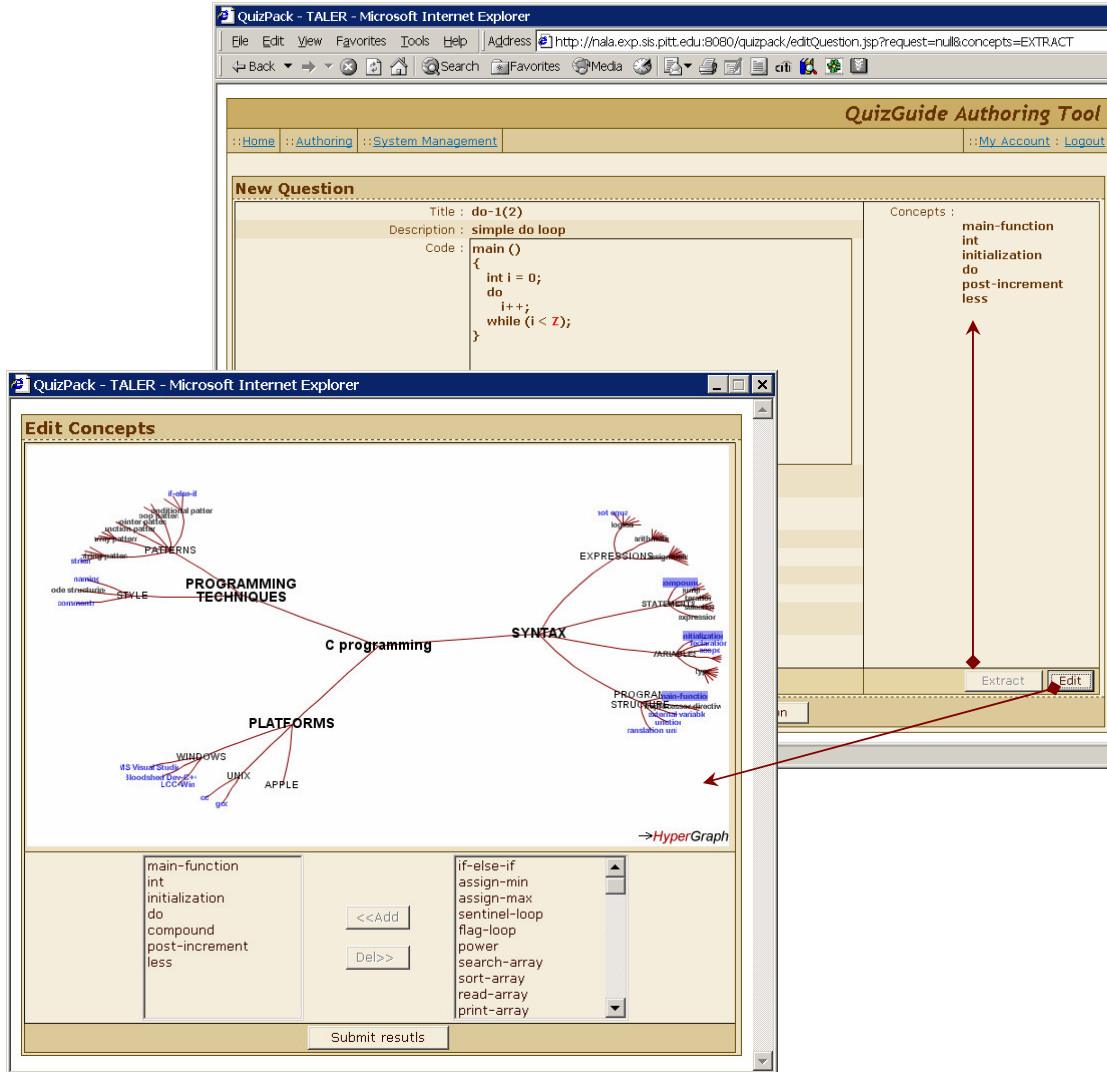


Figure 1. Concept Elicitation from the Code of a Sample Question.

In other words, our intention was to develop a system which supports the authoring of intelligent content according to a teacher's preferences while maximally facilitating this process, but not impose an outside vision of the domain. To ensure this degree of flexibility, our system provides the author with a supplementary interface for editing the extracted list of concepts, or s/he may even create this list from scratch. To start this process an author needs to click on the button *Edit* in the *Concepts* section of the interface. A window loads, where an author can add or remove concepts from the index, either by using the lists of elicited (left) and available (left) concepts or by browsing the domain ontology.

The developed ontology of C programming contains about 150 concepts. About 30 of them are meta-concepts; their titles are written in black font. An author cannot add meta-concepts to the index and may use them only for navigational purposes. Leaves of the

ontology can be either in the index or in the list of available concepts. First, they are represented by blue font on the white background, second, they are written in the light-blue squares. By clicking on leaves of the ontology an author adds (or removes if had already been added) a corresponding concept to the index: the background of the node in the ontology is changed and the concept moves from one list to another. The set of ontology leaves is a superset for the number of concepts available for automatic extraction. Figure 1 demonstrates the process that happens when an author wants to add a concept to the generated list. The parsing component has identified the concept "main-function" in the code of sample example. The compound operator is syntactically a part of the function definition, though the parser has not identified it as a separate concept. However, a teacher might want to stress that this is a particular case of compound operator and add this component by hand. As you can see, the index lists on the main window and on the window of the manual concept elicitation are different. The concept "compound" is added to the index manually, but is not saved at the moment. Hence, an author has freedom: s/he can choose to rely on the automatic indexing or can perform more precise manual indexing that best fits her/his needs.

As an ontology visualization tool we use the hypergraph software (<http://hypergraph.sourceforge.net/>), which provides an open source easy-tuneable platform for manipulating hyperbolic trees [15]. A number of research and practical projects are conducted currently on different types of tools for the visualization of large concept structures [16; 17]. Hyperbolic trees allow one to shift the focus away from unnecessary information while preserving the entire structure of the tree (or its sufficient part) on the screen. Since, our choice of ontology type is a simple taxonomy, tree structure is the best choice for representing the relationships of the domain concepts and organizing them into helpful navigational components.

3. Prerequisite/Outcome Identification

The outcomes of the concept elicitation stage are concept lists for all content elements (in this case, questions). However, prerequisite-based adaptive navigation support technique that we apply [7] requires all concepts associated with a content element to be divided into prerequisite and outcome concepts. Prerequisites are the concepts that students need to master before starting to work with the element. Outcomes denote concepts that are being learned in the process of work with the element.

We use an original algorithm for the automatic identification of prerequisite and outcome concepts for each element. This algorithm is also collaborative because it takes into account a specific way of teaching the course provided by the instructor. The source of knowledge for this algorithm is a sequence of learning goals defined by the instructor [18]. Each goal typically represents a course lecture. To define a new goal an instructor simply needs to group together all content elements that support a specific lecture. The result of this process is a sequence of groups of content elements that corresponds to a course-specific sequence of lectures. The prerequisite/outcome separation algorithm starts with the first lecture and works iteratively through the sequence of lectures.

- All concepts associated with content elements that form the first group (first lecture) are declared the outcomes of the first lecture and are marked as outcomes in the index of all content elements that form the first group.
- All concepts associated with content elements that form the second group (second lecture) are divided into lecture outcomes and lecture prerequisites. All concepts already listed as outcomes of the first lecture are defined as prerequisites of the second lecture. They are marked as prerequisite concepts for each content element in the second group. The concepts that were first mentioned in the second group become

outcomes of the second lecture. They are marked as outcome concepts for each content element in the second group.

- This process is repeated for each following group. On each step we separate concepts that are newly introduced and concepts that were introduced in one of the earlier lectures. The result of the process is a separation of prerequisite and outcome concepts for each lecture and each listed content element. A by-product of this process is the identification of the learning goal (a set of introduced concepts) of each lecture. Note that for each concept there is exactly one “home lecture” that introduced this concept.

Once the content elements are indexed and the goal sequence is constructed, any future additional element can be properly indexed and associated with a specific lecture in the course. The element is to be 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 again that the outcome identification is adapted to a specific way of teaching a course, as it is *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. Naturally, content sequencing in a course should be adapted to the instructor's preferred method of teaching. This is in contrast to the case when a teacher willing to use an adaptive system with the side-authored content in the class is forced to adjust the course structure to the system's view on it, or more precisely, to the view of the authors of the system.

4. Discussion and Future Work

This paper focuses on a new generation of authoring tools that support teachers as authors on intelligent content. We have presented a specific authoring system for automated collaborative indexing of parameterized questions. Although, some part of the system (the described automated approach to concept extraction, using a parsing component), is specific for the learning content based on the programming code (questions and code examples), we believe that the proposed general idea is applicable for a broad class of domains and content types. In less formalized domains, where concepts do not have a salient grammatical structure, the classic information retrieval approach could be used instead of parsing. The other two key ideas: ontology-based authoring support and prerequisite-outcome identification are domain independent.

The presented approach to intelligent content authoring as well as the implemented interface need exhaustive evaluation. Several research questions may arise:

- Does the proposed algorithm for prerequisite/outcome identification and concept elicitation provide good source for adequate adaptation?
- How helpful will the approach and the tool be for an arbitrary teacher, in indexing her/his own content?
- Are authors going to use the manual concept elicitation or will they stick to the automatic indexing? In the former case, will they prefer ontology-based authoring or simply turn to list manipulation?
- Are teachers going to take the time to author the adaptive content?

At the moment of writing we have formally evaluated one interactive component of the system – the concept-indexing tool based on hyperbolic trees. This component was evaluated in the context of a different authoring tool - Collaborative Paper Exchange [19]. The users of this tool are required to write summaries of research papers and index each summary with domain concepts. A short study presented in [19] evaluated the usability of the tool and compared two approaches to ontology-based indexing – traditional approach based on list selection and hyperbolic tree indexing. While the study showed that the current version of

hyperbolic tree indexing is far from perfection, nine out of 14 subjects preferred hyperbolic tree indexing over traditional list-based indexing.

We will continue the evaluation process using several interactive tools we have developed for different types of learning activities. Our ultimate goal is to involve teachers into practical use of these tools and perform both subjective analysis of usability and objective evaluation of the labor-intensiveness of adaptive instruction authoring.

References

- [1] Brusilovsky, P. and Peylo, C. Adaptive and intelligent Web-based educational systems. *International Journal of Artificial Intelligence in Education*, 13, 2-4 (2003), 159-172.
- [2] 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.
- [3] 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/>.
- [4] Koedinger, K.R., Anderson, J.R., Hadley, W.H., and Mark, M.A. Intelligent tutoring goes to school in the big city. In: Greer, J. (ed.) *Proc. of AI-ED'95, 7th World Conference on Artificial Intelligence in Education*, (Washington, DC, 16-19 August 1995), AACE, 421-428.
- [5] Arroyo, I., Schapira, A., and Woolf, B.P. Authoring and sharing word problems with AWE. In: Moore, J.D., Redfield, C.L. and Johnson, W.L. (eds.) *Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future*. IOS Press, Amsterdam, 2001, 527-529.
- [6] Murray, T. Authoring Intelligent Tutoring Systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10 (1999), 98-129, available online at http://cbl.leeds.ac.uk/ijaied/abstracts/Vol_10/murray.html.
- [7] Brusilovsky, P., Eklund, J., and Schwarz, E. Web-based education for all: A tool for developing adaptive courseware. *Computer Networks and ISDN Systems*. 30, 1-7 (1998), 291-300.
- [8] De Bra, P. and Calvi, L. AHA! An open Adaptive Hypermedia Architecture. *The New Review of Hypermedia and Multimedia*, 4 (1998), 115-139.
- [9] Henze, N. and Nejdil, 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.
- [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, May 29-31, 2002), 376-379.
- [11] Sosnovsky, S., Shcherbinina, O., and Brusilovsky, P. 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, November 7-11, 2003), AACE, 309-316.
- [12] 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.
- [13] Barr, A., Beard, M., and Atkinson, R.C. The computer as tutorial laboratory: the Stanford BIP project. *International Journal on the Man-Machine Studies*, 8, 5 (1976), 567-596.
- [14] Weber, G. Individual selection of examples in an intelligent learning environment. *Journal of Artificial Intelligence in Education*, 7, 1 (1996), 3-31.
- [15] Lamping, R. and Pirolli, P. A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. In: Katz, I., Mack, R. and Marks, L. (eds.) *Proc. of CHI'95*, (Denver, May 7-11, 1995), ACM, 401-408.
- [16] Uther, J. and Kay, J. VIUM, a Web-based visualization of large user models. In: Brusilovsky, P., Corbett, A. and Rosis, F.d. (eds.) *User Modeling 2003. Lecture Notes in Artificial Intelligence*, Vol. 2702. Springer Verlag, Berlin, 2003, 198-202.
- [17] Ziegler, J., Kunz, C., Botsch, V., and Schneeberger, J. Visualizing and exploring large networked information spaces with Matrix Browser. In: *Proc. of 6th International Conference on Information Visualisation, IV'02*, (London, UK, July 10-12, 2002), IEEE Computer Society.
- [18] Brusilovsky, P. 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*. Dordrecht, Kluwer, 2003, 377-409.
- [19] Yudelson, M. and Brusilovsky, P. Collaborative Paper Exchange. In: *Proc. of World Conference on E-Learning, E-Learn 2005*, (Vancouver, Canada, November, 2005), AACE, submitted.