A User Modeling Server for Contemporary Adaptive Hypermedia: an Evaluation of the Push Approach to Evidence Propagation

Michael Yudelson, Peter Brusilovsky, Vladimir Zadorozhny

School of Information Science, University of Pittsburgh
135 N. Bellefield Ave. Pittsburgh, PA 15232, USA
mvy3@pitt.edu,{peterb, vladimir}@sis.pitt.edu

Abstract. Despite the growing popularity of user modeling servers, little attention has been paid to optimizing and evaluating the performance of these servers. We argue that implementation issues and their influence on server performance should become the central focus of the user modeling community, since there is a sharply increasing real-life load on user modeling servers, This paper focuses on a specific implementation-level aspect of user modeling servers – the choice of *push* or *pull* approaches to evidence propagation. We present a new push-based implementation of our user modeling server CUMULATE and compare its performance with the performance of the original pull-based CUMULATE server.

1 Introduction

User modeling servers are becoming more and more popular in the field of user modeling and personalization. The predecessors of the present user modeling servers, known as generic user modeling systems [9; 10], were developed to distill the user modeling functionality of the user models within adaptive systems and to simplify the work of future developers of these systems. Modern Web-based user modeling servers [1; 4; 8; 11; 12; 14] added another important function: to serve as a central point for user modeling and the provision of information about a user in a distributed environment, where several adaptive systems may simultaneously communicate with the same server to report or request information about the user.

Typical usage of a user modeling server follows: an adaptive system interacts with the user and sends the results of that interaction to the user modeling server. In some cases, the user modeling server simply stores the information provided by the adaptive system. For example, the adaptive system can report user age, as provided by the user herself, which will be stored by the server for future use. In other cases, the user modeling server has to make inferences based on the evidence it receives. Typically, inferences are formed when the adaptive system reports some meaningful interaction event (i.e., the user just read a specific news article or solved a specific educational problem), which is then distilled into meaningful user parameters such as user knowledge or interest. The information about the user accumulated by the server can further be requested by various adaptive systems that are also working with this user. While the main function of a modeling server is to answer requests about stored or derived user parameters (such as age, knowledge, or interests) some modern servers such as Personis [8] and CUMULATE [4] are also able to respond to different requests about the history of the user's interactions.

With an increasing number of adaptive systems accessing the same server and the increasing complexity of user model inferences, the performance of a user modeling server is becoming an important factor. However, with the exception of the pioneer work of Kobsa and Fink [11], the literature on user model servers focuses solely on the conceptual architectures and functionality without paying any attention to implementation details and real-life performance. We argue that these issues should receive more serious attention from the user modeling community. As our experience shows, a range of implementation details may dramatically affect the server performance. A specific implementation aspect that is discussed in this paper is the balance between the *push* and *pull* styles of inference that is chosen within user model servers. A server with *pull* inference deduces user parameters (such as knowledge or interests) from collected observations "on demand" – i.e., when requested. A server with *push* inference updates user parameters after each reported observation, thus keeping them instantly available. While both approaches may be used to implement the same conceptual architecture, the choice of approach may determine the ultimate productivity of the server, depending on the individually required balance of reports and requests to the server.

Historically, in several kinds of adaptive systems that build a model of user knowledge (such as intelligent tutoring systems), event reports are frequent while user model requests are rare. For example, after a good number of reported user events, created during the process of solving a problem or the exploration of a virtual lab, the system comes to a decision point, where information about a user is required, such as to choose the next task to solve. Hence, the issue of response delay to read requests hasn't been considered as a critical issue. Read request response time becomes crucial when the following conditions are met:

- user models become more complex,
- more users start using the adaptive systems more frequently, hence increasing the volume of data sent to the user model, and
- the user model is queried for updated information about the user more often.

When these three conditions are met, the propagation of evidence starts to cost a lot more when it's done only upon read request as opposed to being done right after the arrival of new evidence.

Recently, we have witnessed the above situation arise in our research. Our pull-propagation user modeling server CUMULATE [4] was originally able to accommodate a small set of adaptive educational activities, which was used by a small group of students (20-30 people). Over the years, with the growth of the number of adaptive applications, the number of users, and the frequency of their work with the system [3] we started to experience noticeable delays when querying user parameters. After several semesters, the delays had become unacceptable (up to 5-7 seconds per each request). We have attempted to introduce a pseudo-optimization to reduce the inference load caused by the user model read requests by introducing the concept of query precision. Precision became an additional parameter in a read request to the user model. It specified how ‘fresh’ the user model was required to be. If the last state of the user model was calculated less than the specified amount of time ago, then the current state of the user model was considered acceptable and was reported without additional inference. This pseudo-optimization didn’t help. As it turned out, each of our adaptive applications demanded ‘fresh’ data from the user model *after each reported event*. For example, QuizGuide and NavEx [5], two adaptive hypermedia services, attempted to update the state of link annotation after every user action (such as answering a question or accessing an example line of code). Since a fresh read of the user model was required after every click of every user, this resulted in a large volume of user model requests, which caused unacceptable delays. Our analysis of contemporary work on adaptive hypermedia demonstrated that the same need to regenerate adaptive annotations after each click is shared by many systems which use adaptive link annotation and this caused us to design a new version of CUMULATE that can support a large number of users working with contemporary adaptive hypermedia.

Given the increased volume of read requests, we decided that one of the main reasons for the original CUMULATE performance problems was the use of pull evidence propagation on the implementation level. To resolve these problems we developed a new version of our user modeling server – CUMULATE 2 – which introduced push evidence propagation. The CUMULATE 2 server was successfully used for two semesters and its performance evaluation returned positive results. This paper reports our work on CUMULATE 2 and is organized in the following way. Section 2 presents the conceptual architecture implemented by both the original CUMULATE and CUMLATE 2 user modeling servers. Section 3 provides details about the implementation of the evidence propagation in each of these servers. Section 4 reports the comparative evaluation of the two servers. Finally, we conclude with section 5.

2 The Conceptual Architecture of a User Modeling Server

How does a typical user modeling server (UMS) works? It receives reports of the user’s activities from external applications (i.e., links the user has followed, pages read, questions answered, etc.). From these reported activities, the UMS infers user parameters such as knowledge or interests. The inference is typically based on some kind of knowledge about how each user action contributes to the change in user knowledge, interests, or other parameters. Inference is done using various approaches, ranging from simple ad hoc math to Bayesian Networks [6] and ontology reasoning [7].

A typical approach to connecting actions with user model parameters in educational adaptive hypermedia is called ‘indexing.’ Educational content is indexed with metadata created beforehand (ontology, taxonomy or flat list) or extracted from content itself using machine learning methods. The indexing is done manually by teacher or semi-automatically with the help of an intelligent parser. Chunks of domain knowledge are referred to as keywords, concepts or topics, depending on the granularity and method of extraction. In simple cases, each piece of content is connected to one chunk of domain knowledge. For example, in QuizGuide [3] – a system that serves parameterized in the domain of the C- programming language – each quiz is assigned to one topic. In other cases, each piece of content is assigned to a set of chunks. For instance, in the system NavEx [13], which provides dissected code examples, each example is indexed with a set of domain concepts.

The two UMS discussed in this paper – the original CUMULATE (which we will call *legacy* CUMULATE, to avoid confusion) and the newer CUMULATE 2 – are typical representatives of a large class of centralized educational user modeling systems. Both of them implement the same conceptual architecture for centralized user modeling that we summarize below.

A user modeling server stores or uses information about the following data objects:

- users,
- groups of users,
- learning objects, and
- domain concepts.

The corpus of learning objects is comprised of several sets of learning objects that are supplied by external applications. For example, learning objects could be parameterized online quizzes or dissected program examples. Domain concepts, contained in the *metadata* corpus, consist of a number of domain ontologies (represented as hierarchies, networks, or flat lists) that are called upon to describe learning objects in terms of knowledge components (often referred to as concepts or sometimes topics). For instance, in the domain of programming language knowledge, components might include such concepts as 'arithmetic operations,' 'addition,' 'data structure,' 'array,' etc. Listed objects are linked by the following relations:

- Group-user membership links. User groups consist of several users and users can be members of several groups.
- Links between learning objects allow learning objects to aggregate subordinates. Leaf objects do not necessary have to be invoke-able but user activity can be attributed to them. For instance, a learning object 'quiz' could consist of several 'questions.'. Both quiz and question can be invoked. A learning object, such as 'code example,' could consist of several 'lines of code.'. In this case, lines are only invoked as a part of the whole code example. These links are optional.
- Links of diverse types connect knowledge components within domain ontologies. For example, 'arithmetic operations' and 'data structure' would be parents to 'addition' and 'array.' These links are optional as well.
- 'Indexing' links between knowledge components and learning objects. These links are crucial for the user modeling process, since they allow the user model to 'propagate' the results of user activity with learning objects, in order to create knowledge components and make assertions about the user mastery of those components. For instance, the line of code '`for(int i=0; i<10; i++)`' (as part of a code example or part of question of a quiz on C - programming language) could be associated with the knowledge components 'loops,' 'for-loop,' 'declaration of a variable,' 'arithmetic expressions,' 'post-increment,' etc.

There are two more special types of relationships in our user model. The first one is evidence links, which describe the results of user activity. They link learning objects to users and groups (because users interact with learning objects as members of some group). Evidence links are assigned timestamps and contain results of such interaction. Usually, the result is expressed in the form of a decimal value between 0 and 1, with 0 denoting an unsuccessful result and 1, the opposite.

The second special type of link, assertions about user knowledge – represent the user model's probabilistic hypotheses about the user knowledge level of some knowledge components. Assertions are modeled with respect to the cognitive levels of Bloom's Taxonomy [2].

Propagation of evidence about user knowledge is driven by reports of user activity from external applications. These reports are generated when a user, for example, clicks on one line of a dissected code example or answers one question of an online quiz. A set of inference agents [4] are configured to aggregate incoming evidence and infer the user's knowledge of concepts belonging to domains stored in the user model based on evidence of user work with various sets of learning objects supplied by specific external application(s). Agents propagate evidence from events to knowledge components of the user model by using indexing links between the learning objects that generated the evidence and knowledge components. The path that evidence travels is shown in Fig. 1. It is important to note that the presented framework is relatively universal. While in our case it was applied to user knowledge modeling, similar approaches have been used for modeling user interests and other features.

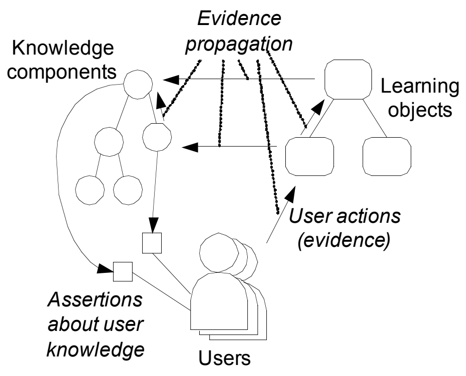

Fig. 1. The structure of the user model, showing the path of evidence propagation

The conceptual description above gives a structural framework and doesn't suggest any particular implementation of the user modeling server's internal inference mechanisms. The inference agents can be implemented using Bayesian Networks, machine learning, or information retrieval methods. One aspect of inference implementation is considering when such inference happens. Possible options include the *pull* approach, where inference is done 'just-in-time,' after a request for inferred information has been received. In other words, external applications *pull* assertions about the user out of the UMS. An alternative to *pull* is the *push* approach, where the computation of user knowledge is done upon arrival of new evidence that *pushes* itself through the user model from the learning objects to the knowledge components.

In this paper, we draw a comparison of two user modeling servers: one implementing pure pull strategy of inference, and the other implementing the push strategy. The following sections describe implementation details for both of them.

3 User Modeling in Legacy CUMULATE and CUMULATE 2

Legacy CUMULATE [4] is a centralized user modeling server that implements pure pull approach. Here, inference agents are not activated by the arrival of new evidence (such as a write operation to the user model). As new evidence arrives, it is constantly recorded in the event history and is not aggregated until an external application requests information about the user's knowledge (a read query to the user model).

Inference in legacy CUMULATE is performed by a set of SQL queries to the UMS database. The process of evidence aggregation is implemented by nesting queries. Because of the just-in-time nature of evidence propagation in the legacy CUMULATE, as our evidence store size increased we began to experience proportionate delays in response to user model read requests. In addition to the growth of evidence, storing new adaptive applications demanded more complex models. Instead of indexing learning objects with a single domain topic (a rather coarse-grained chunk of the domain), we have switched to indexing them with a set of finer -grained concepts. The increased knowledge -component -to -learning -object -ratio, in addition to growth of the event base has slowed the inference process.

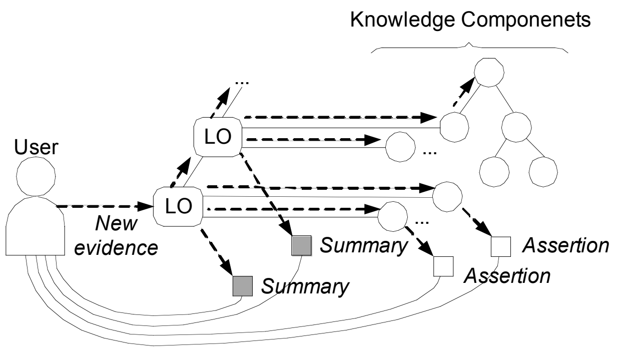

Fig. 2. Propagation of evidence in CUMULATE 2

In our second attempt to implement the conceptual architecture described above, we decided to switch from pull to push inference of user knowledge, in order to improve performance of the user modeling server. Our new UMS CUMULATE 2 performs the inference of user knowledge immediately after arrival of new evidence. The evidence log is used as a backup in case of server failure, when upon restart, CUMULATE 2 sequentially propagates all evidence cached in the log in the same fashion evidence is propagated in the working mode. The CUMULATE 2 propagation architecture can be used with a range of *incremental* user modeling approaches (i.e., where new values for knowledge, interests or other features can be determined by combining old values with new evidence). The current inference agents in CUMULATE 2 use a set of threshold, averaging and asymptotic formulas for evidence propagation. For example, user knowledge of the concept grows asymptotically (on a transposed cubic curve) each time a user successfully answers one question of a quiz which is related to this concept. However, the architecture allows the use of Bayesian inference approaches such as used in SMODEL [14] and other Bayesian user modeling systems.

CUMULATE 2 is implemented as a network of interactive Java objects. A single entry point to the server API is created in the form of an instance singleton class. Java servlets further abstract the server's API via an HTTP interface. External applications can send a write request to the servlet that is responsible for UMS updates with parameters of a single piece of evidence about its user. When a new piece of evidence arrives, it is first checked for consistency of server settings (existence of the user and user group with such user, identity of a reporting application, existence of the learning object that the user is reported to be interacting with). Second, evidence is stored in the database. Third, the piece of evidence is propagated throughout the user model.

Results of evidence propagation in the form of summaries and assertions are cached for faster access (Fig. 2). Each learning object summarizes evidence that 'passes' through it by counting the total number of pieces of evidence, mean result value (i.e., number of correctly answered questions over all question attempts). Then the learning object passes the evidence to its superiors (e.g.., question to quiz, or individual code line to a full dissection) and to the knowledge components it has been indexed with. Superior learning objects aggregate the 'count' of pieces of evidence by summing the counts of their subordinates, and find the mean interaction result by taking the average of the mean interaction results of subordinates.

Each knowledge component aggregates evidence by computing the probability of a user mastering it. The formulas for computing these probabilities are configured individually for external adaptive applications. For instance, knowledge components aggregate evidence coming from users browsing dissected code examples [13] by applying an ad hoc step function that sets the threshold of 10 'clicks' on annotated lines of code that connect to the knowledge component as the amount of interaction which will enable the user to master this knowledge component. If the user has made less than 10 clicks, then the probability is taken as the number of clicks made over 10, and 1 otherwise. These probabilities are recorded in the slot that corresponds to Bloom's 'comprehension' cognitive level.

Evidence from learning objects that represent questions of online quizzes [3] are aggregated using a sigmoid asymptotic function. Probability of the user mastering a knowledge component grows with each successful answer to the quiz question. The first two to three attempts to successfully apply the knowledge components result in the slow growth of the probability of mastery (a warm-up period), further success results in the linear growth of probability and as probability approaches 1, the increments asymptotically decrease.

Queries to CUMULATE 2 for the snapshot of an individual user model are handled by another servlet – the report manager. At this point, CUMULATE 2 performs a simple lookup operation and responds with an XML document that describes the requested information.

4 A Preliminary Evaluation

Over the several years that we have been using the legacy CUMULATE as our primary UMS, we have accumulated a large number of records. Records of our most heavily used and researched application, QuizPACK, contain about 19,000 pieces of evidence obtained in more than 13,000 sessions. In a typical session, users answered 28 questions. On average, users generated a single piece of evidence within a session once every 102 seconds.

Using these figures as a 'realistic' baseline, we compared the performance of the legacy CUMULATE to the performance of CUMULATE 2 to see whether the shift from pull to push evidence propagation strategy made any difference. Since our main reason behind the strategy switch was to overcome large read request delays, we were primarily interested in whether the situation improved in CUMULATE 2 (i.e., whether the delay became smaller). Our secondary point of interest was whether the write request delay grew larger for CUMULATE 2, since CUMULATE 2 performs more computations when updating the user model, while legacy CUMULATE doesn't compute anything at that point.

A small experiment was setup, where we subjected both versions of the user modeling server to various types of loads. Both servers were configured identically. The size of the learning objects corpus was 1000, while the metadata corpus was 500. The ratio of knowledge concepts to each learning object ranged from 5 to 100. Servers were running on the same software/hardware.

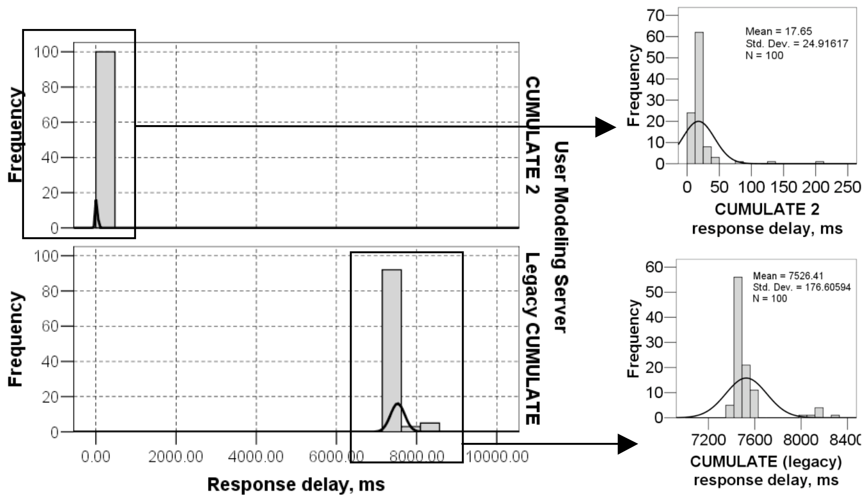

Fig. 3. Comparing the read request delays of CUMULATE (bottom) and CUMULATE 2 (top)

To quantitatively compare the servers' ability to handle read requests from external applications we sent 100 consecutive queries to each of them. Fig. 3 shows that CUMULATE 2 wins a convincing victory with 18 milliseconds average response time over the legacy CUMULATE, which delays responses to read requests for 7526 milliseconds. The left side shows histograms of the read request delays for legacy

CUMULATE (bottom) and CUMULATE 2 (top) placed on one scale. Call-outs on the right show the histograms in greater detail.

We have also investigated the ability of the servers to handle write requests. We varied server loads from 1 second to 80 milliseconds between requests for a duration of 3,000 milliseconds. At the peak load of 80 milliseconds between write requests, legacy CUMULATE was able to complete 90% of the requests within 32 milliseconds. Under these conditions, CUMULATE 2 was only able to complete 90% of requests within 126 milliseconds.

As we have mentioned above, when users employ our tools for an introductory programming course, they typically answer one quiz question per 102 seconds during a learning session. For each update of the user mode, the user expects an update of the user model (expressed as changed annotations for quizzes and questions).

In this situation, CUMULATE 2 is able to support roughly 700 users working simultaneously, namely, 126 milliseconds per write request and 18 milliseconds per read, giving us 144 milliseconds for the write-read cycle. Knowing that user answers come once in 102 seconds we have 102 * 1000 / 144 = 708 ≈ 700. This is more than enough, given that the size of the class is rarely over 20 students. Taking into account that at any moment no more than 25% of students' sessions overlap, CUMULATE 2 could easily support a user population that is 4 times as large. The legacy CUMULATE is quite slow because of read requests' delays, even when only one student is working with the adaptive applications that use the server.

This shows us that moving from pull to push propagation did in fact pay off and the improvement is quite significant.

5 Conclusions

In this paper we have described our user modeling server CUMULATE 2, which implements the push approach to evidence propagation. We have drawn initial comparisons between CUMULATE 2 and our legacy user modeling server CUMULATE, which implements the pull evidence propagation strategy.

Results of the comparison show that switching from pull to push propagation has dramatically decreased query delays to the user modeling server (from 7526 to 18 milliseconds). The fact that the propagation strategy was the only tangible difference between the two servers allows us to conclude that it is the push propagation that has caused the performance leap. However, it is only a preliminary result. Both write and read requests to the user modeling servers were quite simple, namely, 'update with one piece of evidence' and 'read full user model.'. Detailed investigation is needed to understand how environment and internal conditions as well as parameters of the requests influence the performance of the servers. We intend to continue analysis of the proposed method with the twin goals of understanding underlying factors that influence its performance and building a detailed cost model of the evidence propagation process.

Acknowledgments. This material is partially based upon work supported by the National Science Foundation under Grants No. 0310576 and 0426021.

References

1. Agostini, A., Bettini, C., Cesa-Bianchi, N., Maggiorini, D., and Riboni, D.: Integrated Profile Management for Mobile Computing. In: Proc. of Workshop on Artificial Intelligence, Information Access, and Mobile Computing at IJCAI'2003, Acapulco, Mexico (2003), http://www.dimi.uniud.it/workshop/ai2ia/cameraready/agostini.pdf
2. Bloom, B. S.: Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain. David McKay Co Inc., New York
3. Brusilovsky, P. and Sosnovsky, S.: Engaging students to work with self-assessment questions: A study of two approaches. In: Proc. of 10th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE'2005, Monte de Caparica, Portugal, ACM Press (2005) 251-255
4. Brusilovsky, P., Sosnovsky, S., and Shcherbinina, O.: User Modeling in a Distributed E-Learning Architecture. In: Ardissono, L., Brna, P. and Mitrovic, A. (eds.) Proc. of 10th International User Modeling Conference, Berlin, Springer-Verlag (2005) 387-391
5. Brusilovsky, P., Sosnovsky, S., and Yudelson, M.: Addictive links: The motivational value of adaptive link annotation in educational hypermedia. In: Wade, V., Ashman, H. and Smyth, B. (eds.) Proc. of 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'2006), Dublin, Ireland, Springer Verlag (2006) 51-60, available online at http://dx.doi.org/10.1007/11768012_7
6. Bunt, A. and Conati, C.: Probabilistic student modelling to improve exploratory behaviour. User Modeling and User Adapted Interaction **13**, 3 (2003) 269-309
7. Conlan, O., O'Keeffe, I., and Tallon, S.: Combining adaptive hypermedia techniques and ontology reasoning to produce dynamic personalized news services. In: Wade, V., Ashman, H. and Smyth, B. (eds.) Proc. of 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'2006), Dublin, Ireland, Springer Verlag (2006) 81-90
8. Kay, J., Kummerfeld, B., and Lauder, P.: Personis: A server for user modeling. 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), Málaga, Spain (2002) 201-212
9. Kobsa, A.: Generic user modeling systems. User Modeling and User Adapted Interaction **11**, 1-2 (2001) 49-63
10. Kobsa, A.: Generic user modeling systems. In: Brusilovsky, P., Kobsa, A., Neidl, W. (eds.): The Adaptive Web: Methods and Strategies of Web Personalization. Lecture Notes in Computer Science, Vol. 4321. Springer-Verlag, Berlin Heidelberg New York (2007)
11. Kobsa, A., Fink, J.: An LDAP-based User Modeling Server and its Evaluation. User Modeling and User-Adapted Interaction **16**, 2 (2006) 129-169
12. van der Sluijs, K. and Houben, G.-J.: Towards a Generic User Model Component. In: Proc. of PerSWeb'05, Workshop on Personalization on the Semantic Web at 10th International User Modeling Conference (2005), available online at http://www.win.tue.nl/persweb/Camera-ready/13-Sluijs-full.pdf
13. Yudelson, M. and Brusilovsky, P.: NavEx: Providing Navigation Support for Adaptive Browsing of Annotated Code Examples. In: Looi, C.-K., McCalla, G., Bredeweg, B. and Breuker, J. (eds.) Artificial Intelligence in Education: Supporting Learning through Intelligent and Socially Informed Technology. IOS Press, Amsterdam (2005) 710-717
14. Zapata-Rivera, J.-D. and Greer, J. E.: SMODEL Server: Student modeling in distributed multi-agent tutoring systems. 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) 446-455