8th World Conference on Artificial Intelligence in Education (AI&ED '97), August 20-22, 1997, Kobe Japan, pp. 55-62.

An Architecture for Intelligent Collaborative Educational Systems

Dan Suthers and Dan Jones

Learning Research and Development Center, University of Pittsburgh
3939 O'Hara Street, Pittsburgh, PA 15260
advlearn+@pitt.edu
http://advlearn.lrdc.pitt.edu/

Abstract: A major technological concern of our work is to improve the cost effectiveness, reusability, and interoperability of advanced educational software. To make these technologies viable, we must be able to add component functionality incrementally, and enable systems to interoperate with commercial software and internet resources. We have designing and implemented an architecture that places shared resources and "heavyweight" functionality on servers, and uses Java and Netscape to deliver student interfaces on a wide variety of client platforms at any location with internet access. This paper describes the architecture at five levels of description. Its strengths and weaknesses provide a case study in how to improve the deployability and interoperability of knowledge-based educational software without sacrificing advanced functionality.

1. Introduction

Knowledge-based educational software, such as intelligent tutoring systems, have historically been large, self-contained programs with specialized platform requirements. To make these technologies viable, we must be able add component functionality incrementally, and enable systems to interoperate with commercial software and internet resources [1, 6, 7]. To reduce the cost of materials prepared by developers, and to enable greater collaboration between users, representations of educational materials should be shareable between diverse applications across the internet. Interoperability and reuse considerations suggest a "lowest common denominator" approach, yet we do not want to limit support for more advanced functionality such as domain-specific coaching.

To address these concerns, we have designed an architecture that places shared resources and "heavyweight" functionality on servers, and uses Java and Netscape to deliver student interfaces on a wide variety of client platforms at any location with internet access. The representations used build on existing standards, embedding semantic annotations that support advanced functionality in materials that are also accessible to more conventional software. The implemented system includes groupware and associated tools that support students engaged in critical inquiry processes, such as investigating a scientific problem:

- A collaborative inquiry database that students use to keep track of their inquiry process, including a statement of the problem, hypotheses that have been proposed and evidence offered for and against them, as well as references to information resources and experimental records.
- A Java-based "inquiry diagram" interface to this database, which helps students visualize the important ideas in a debate as concrete objects that can be pointed to, linked to other objects, and discussed;.
- HTML-based interfaces to the database, to enable access when Java tools are not available and to support tabular and textual views on the record of inquiry.
- A coach that is designed to stimulate students' contributions to the critical inquiry process.
- HTML-based reference materials that are structured to scaffold the critical inquiry process, and annotated to support coaching.

The system, called "Belvedere," is a complete redesign and reimplementation of one by the same name described in [12, 13]. The new system has been deployed in four high-schools in the Department of Defense Dependent Schools (DoDDS) overseas. It is currently under evaluation in those sites as well as in our lab.

This paper describes the architecture underlying the Belvedere system, using the architecture as a case study in how to improve the deployability and interoperability of knowledge-based educational software without sacrificing advanced functionality. As an expository device, we use four levels of description for software systems proposed by Frank Belz and David Luckham (personal communications): Interface Presentation, Concepts of Operations, Abstract Implementation, and Resource. In analyzing our own work we have found it useful to begin with a fifth level of description, Concepts of Application, that is independent of the software. This is necessary for design and evaluation with respect to intended objectives. Along with Belz and Luckham, we claim that clarity about level of description helps avoid misunderstandings due to talking at different levels, and enables one to choose to use an existing architecture at one level while rejecting or changing it at another level.

Each of the following sections begins with a general characterization of the corresponding level of description, followed by an informal description of our application or architecture at that level, and a summary of mappings to other levels of description. At each level we discuss reusability and interoperability concerns, and the advantages and disadvantages of our design. The paper concludes with a discussion of further work, both our own and work needed in the AI&ED community.

2. Concepts of Application

At the level of concepts of application, one begins by describing the application domain largely in its own terms (as practitioners view it), and the educational objectives or other task objectives. Then, through cognitive task analysis or other methodology, one identifies barriers to these objectives, and chooses those which the software might be expected to help overcome.

2.1. Critical Inquiry in Science

The Belvedere application domain is learning critical inquiry skills, particularly in science. Since the focus of this paper is on viable architectures rather than this specific application domain, we describe the application only enough to provide background for subsequent discussion. Basic actions of learning critical inquiry in science include

A1. Familiarizing oneself with a field of study
A2. Identifying a problem of interest
A3. Proposing hypotheses (or solutions)
A4. Identifying and seeking evidence that bears on those hypotheses (or solutions)
A5. Drawing conclusions based on the evidence found
A6. Summarizing and reporting the inquiry to others
A7. Evaluating the status of the inquiry, with repeat at any of the steps above
A8. Discussing and coordinating the doing of 1-8 with others.
A9. Obtaining solicited and unsolicited guidance on how to conduct critical inquiry

We identified the following possible barriers to learning critical inquiry in science [12, 13]:

B1. Lack of motivation.
B2. Limited knowledge of scientific domains.
B3. Inability to recognize abstract relationships implicit in scientific theories and arguments about them.
B4. Difficulty keeping track of a complex debate.
B5. Lack of scientific argumentation criteria, and associated biases, e.g., confirmation bias.

We return to elements of both of the above lists in subsequent sections.

2.2. Generality

At the Concepts of Application level, "reusability" is a psychological concern rather than an engineering concern: we must ask how well the task analysis applies to other domains, and hence whether the

pedagogical strategies and forms of scaffolding that are embodied in other levels of the system will transfer well. The generality of our particular analysis is not within the scope of this paper.

3. Interface Presentation

At the interface presentation level, one designs the perceptual/motor experience of the user. Here we describe the functionality available to user in terms of representations of application objects and actions on these objects.

3.1. A Graphical Interface for Critical Inquiry

The Belvedere "inquiry diagram" interface (Figure 1) can be thought of as networked groupware for constructing representations of evidential relations between statements. It uses shapes for different types of statements and links for different kinds of relationships between these statements. Multiple clients can view the same inquiry diagram, with "what you see is what I see" (WYSIWIS) updating. An axillary "chat" window (upper left of Figure 1) supports unstructured natural language communication. Additionally, a software-based "coach" (lower right of Figure 1) provides assistance to students as they engage in their various inquiry activities [5, 14] To avoid interrupting students' thought processes, the coach is minimally intrusive, usually remaining quiet unless students ask for advice, and flashing its light bulb only when it has critical advice to offer. It coaches critical inquiry by asking questions students may not have thought of, based on criteria of inquiry and argumentation in science.

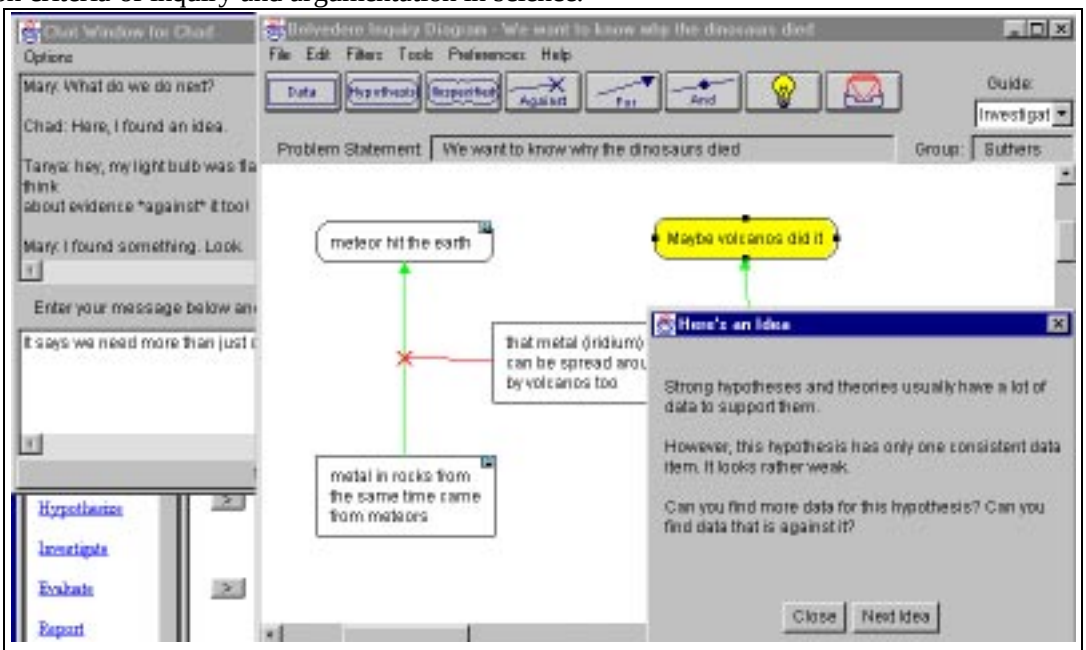


Figure 1. The Belvedere Interface

Belvedere is designed to be used in conjunction with materials presented in a Web browser. The materials are segmented into units at a granularity which a subject matter expert chooses for his or her own inquiry diagrams. "Reference This" buttons in the Web pages enable students to send "references" to these segments into the Belvedere "in-box" (upper right of Figure 1) from where they may be dragged into the inquiry diagram as needed. The small icons in the upper left of each shape indicate that hyperlinks can be followed back to the original document.

3.2. Relations to Concepts of Applications

A well designed interface should support Concepts of Application through a clear mapping of domain objects and actions to interface objects and actions. Furthermore, the interface should address the barriers identified at the superordinate level of analysis, for example by providing visual organizers.

Summarizing from [12,13], here is how the interface is designed to address barriers to learning critical inquiry:

B1. Lack of motivation: Belvedere is designed to support collaborative problem solving, providing peer motivation and engaging activities [4,8 9]. Support for collaboration includes networked WYSIWYS, the chat facility, and the diagram itself, which helps students switch between working independently and working together without losing track of what they are doing.

B2. Limited knowledge of scientific domains: This is addressed in part through on-line materials, and in part through "expert coaches" we are now constructing [14] which can coach based on the knowledge of a particular domain.

B3. Inability to recognize abstract relationships and arguments: Belvedere's diagrammatic representations reify these relationships and make weaknesses and points where further contributions can be made salient [10, 11].

B4. Difficulty keeping track of a complex debate: This is partially addressed by the concrete visual representation, which help students keep track of main points and pending issues.

B5. Lack of scientific argumentation criteria, and associated biases: This is addressed by Belvedere's coach.

Following are some examples of the mapping of Concepts of Application actions to the Interface level:

A1. Familiarizing oneself with a field of study: Browsing the Web materials.

A2. Identifying a problem of interest: Starting a new inquiry diagram, labeled by a problem statement.

A3. Proposing hypotheses: Either selecting the "hypothesis" icon and typing in a statement of a hypothesis, or using a "reference this" button to bring a reference to an existing hypothesis into the diagram.

A4. Identifying and seeking evidence that bears on those hypotheses: The coach helps users identify when evidence is needed. The Web materials themselves along with hands-on activities suggested in those materials provide some sources of evidence. Evidence is recorded as for A3, except the "Data" icon is used.

A5. Drawing conclusions: Belvedere provides a facility for changing and viewing the relative "strength" of the different statements. The coach provides some guidance, but further support is needed here.

A6. Summarizing and reporting the inquiry to others: Currently support is inadequate. Users can print their inquiry diagrams, or convert them into HTML tables that summarize the evidence for and against each major hypothesis.

A7. Evaluating the status of the inquiry, with repeat at any of the steps above: The coach provides some local guidance. Also we provide an outline of phases of activity and a "Guide" menu to help students through these phases.

A8. Discussing and coordinating the doing of 1-8 with others: If not in co-located, users can interact via the Chat window.

A9. Obtaining solicited and unsolicited guidance: The coach provides both.

This analysis has been simplified for this paper: our full analysis specifies the complete interface actions required to carry out each action in the concepts of application.

3.3. Comments on the Analysis

An analysis of this kind has helped us identify some limitations of the Belvedere interface. The mapping is not always clear, and it lacks scaffolding of the overall process. We have begun to address these concerns. An advantage of our approach is that the interface can easily be modified without affecting the other levels of the architecture. As we shall see in the next section, the Interface level of analysis can also be bypassed in favor of a direct mapping of Concepts of Application to Concepts of Operations.

4. Concepts of Operations

At this level one describes how the *software* models the application domain, in terms of classes of objects and the operations that can be performed on them. The specification can take the form of an object-oriented model, or a collection of abstract data types (ADTs).

4.1. Supporting Collaborative Coached Critical Inquiry

To illustrate, below are some objects and operations supported by our system. The numbers in brackets indicate which Concepts of Application actions are being supported.

Inquiry Diagrams. Inquiry diagrams consist of a problem statement, and a collection of statements and relationships between them. The operations abstract communications between the Belvedere interface and a persistent object store. Some of these are New Inquiry Diagram [A2], Open Inquiry Diagram [A2], Add Statement [A3, A4], Add Relationship [A3, A4], Update Statement [A5, A7], and Delete Statement or Relationship [A5] (we retain a complete history of all objects that existed).

Information Search. Accomplished by Get Page [A1] and Send Reference [A3, A4], invoked via the Web browser.

Discussion with Others. A8 is accomplished by Send Message.

Advice Services. Objects include requests, replies, and interruptions; all in support of A9. The client can Request Advice; and the coach can Send Advice, which consists of the advice text and a list of the objects that the advice text refers to. The coach can also send an Interruption, which is a request to perform an interface action that notifies the user that advice is available.

Some important Concepts of Application activities are not supported by this model. These include performing data analysis and visualizations [A4, A5], asking the coach specific questions [A9], and abstracting summaries of the inquiry [A6]. Extensions are being planned to address these concerns.

4.2. Relations to Other Levels

Concepts of Operations supports the User Interface level by providing primitives for creation of, access to, and state changes in objects. Concepts of Operations abstracts from Concepts of Application because the objects or ADTS could be reapplied to other application domains that have similar modeling requirements: a given application is an instance in the class of task domains covered. Hence, Concepts of Operations is the level at which we describe *generic task domains*. A *shell* is a collection of software that applies to a given generic task domain [3].[1] For example, our generic task domain is collaborative critical inquiry with coaching, and our software can be thought of as a shell for such applications.

4.3. Interoperability and Reusability

At the level of Concepts of Operations, interoperability and reusability is aided by shared ontologies. Ontologies are formalized structures (such as hierarchies) that define abstract concepts and the relations between them.[2] The concepts abstract critical features of the particular objects of an application domain. Shared ontologies help people communicate the contents and capabilities of their systems, strategies, etc., for example helping us determine whether the modeling services of a particular piece of software will adequately support our needs in a new application, or whether we can reuse a pedagogical strategy. Shared ontologies also enable us to compose knowledge-based software components because they enable one component to "understand" the contents of data or messages it receives from another component. This is an area we have only begun to explore in our own work, but see [2].[3]

---

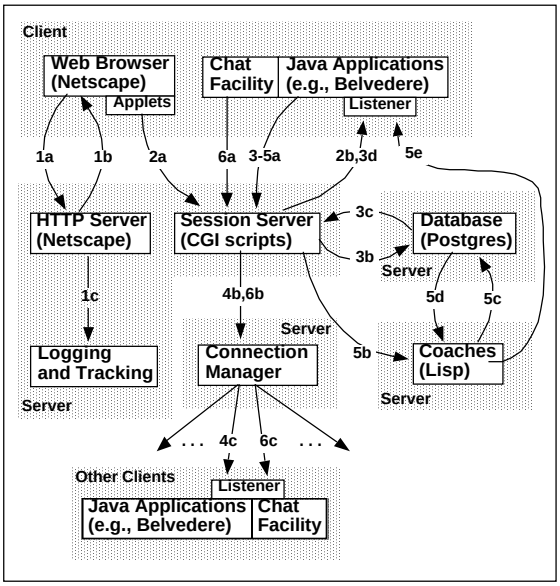
[1] See also http://www.ils.nwu.edu:80/~korcuska/Articles/ITS96Sand/Jona-ITS96.html

[2] See http://WWW-KSL.Stanford.EDU:80/knowledge-sharing/

[3] See also http://advlearn.lrdc.pitt.edu/its-arch/papers/mizoguchi.html

5. Abstract Implementation

At this level one describes the architectural elements and communication between these elements, including software modules such as interpreters, databases, event managers, etc., and data and control flow between them. Figure 2 details our abstract implementation level architecture at the granularity of modules that require network communications. All actions initiated by the user are accomplished via CGI and the response from the CGI call. The decision to use CGI was based upon the availability of the HTTP server (already needed for materials delivery); the ease of interfacing a Java application with the server via the openURL method; and ease of modification and maintenance. Messages for WYSIWIS, coaching, and chat come in asynchronously via a small listener server in the client. The listener runs as a separate thread in the client. The Connection Manager is written in Java. The interfaces are simple and robust: the communication architecture has performed extremely well during our laboratory "stress" testing. Other advantages include portability and low cost (most components are free). A major exception is the Coach, which was implemented in Lisp and Loom for ease of development. The Coach actually consists of several submodules: an argument pattern coach, an expert model coach, and an arbitrator that prioritizes advice from the coaches for presentation based on factors such as discourse history and type of advice [14]. Our architecture enables this use of "heavyweight" environments for advanced functionality, because client platforms need only run Netscape and Java applications. However, we have recently reimplemented the Coach in Java to enable lower cost and portable server delivery.



1. **Browsing** (Get Page)
   a) Client request (HTTP)
   b) Server reply (HTML with embedded Java)
   c) Access logging. (When implemented, Tracker will notify Coaches.)
2. **Referencing On-line Materials** (Send Reference)
   a) Java applet sends reference to server (data embedded in CGI GET)
   b) Reference sent via socket in application specific protocol
3. **Application Requests and Updates** (New Inquiry Diagram, Open Inquiry Diagram, Add Statement, Add Relationship, Update Statement, Delete Statement)
   a) Request or update sent to Session Server (data embedded in CGI GET)
   b) Server queries or updates Database (SQL requests)
   c) Database replies with results or return code
   d) Reply sent to client (response to CGI GET. User was able to continue working before reply received.)
4. **Updates Propagated to Other Clients** (WYSIWIS for events generated in #3)
   a) Update sent to Session Server (subset of 3a)
   b) Connection Manager informed of update (TCP socket; application specific protocol)
   c) Connection Manager formats message and informs all clients that are using the same workspace (TCP socket; application specific protocol)
5. **Coaching** (Request Advice Send Advice)
   a) Update or advice request sent to Session Server (data embedded in CGI GET)
   b) Update or advice request sent to Coach dispatcher (TCP socket; application specific protocol)
   c) Coach queries Database if needed to determine state (SQL, read-only)
   d) Database replies
   e) Coach sends client advice, if requested. Coach sends interrupt when update activated high priority advice (TCP socket; application specific protocol)
6. **Chat Facility** (Send Message)
   a) User's comment sent to Session Server (data embedded in CGI GET)
   b) Session Server sends comment to Connection Manager (TCP socket; application specific protocol)
   c) Connection Manager forwards to users in same workgroup (TCP socket; application specific protocol)

Figure 2: Abstract Implementation Layer

5.1. Relations to Concepts of Operations

Concepts of Operations abstracts functionality from structure in the Abstract Implementation, by indicating which subsets of Abstract Implementation layer are involved in a given functionality (as shown in the lists of Figure 2). Concepts of Operations provides the semantics of communications, and Abstract Implementation provides the syntax and protocol.

5.2. Interoperability and Reusability Issues

Communication is the key to interoperability and reusability at this level. Specifically, the use of standard protocols where they exist facilitates the interchange, addition, or reuse of components. Our current communication protocols and representations are summarized in Figure 2. Some of the advantages have already been discussed, including simplicity, robustness, portability, and low cost. Also, it is easy to add or change clients using CGI scripts. This was not true of the coach described above; however the recently finished Java-based Coach utilizes the same communication protocols (and Java networking code) as other clients. These changes facilitate the easy addition of new coach modules and the distribution of coach functionality across platforms: one can take a client, remove the GUI, and plug in a coach. Furthermore, the architecture permits interaction with other architectures and components. For example, we are currently preparing a MOO-based demonstration in which a simulation by Ken Forbus sends simulation results as Data objects into the Belvedere in-box, and a tutor by Ken Koedinger comments on how these data objects are linked in to the inquiry diagram.

The above design is limited in several ways. Some of the protocols are application specific. This is probably unavoidable; although some reuse may be facilitated by shared ontologies at the Concepts of Operations level. We have begin another cycle of redesign to enable delivery using other databases and other server class machines. Prototype versions of RMI and CORBA server interfaces have been implemented and are currently undergoing testing and evaluation. Our new design will also greatly simplify the addition of new types of clients. (We plan to add clients that manipulate influence diagrams, causal loop diagrams, and concept maps.) Under the new design the protocols are data-driven, so that only minor modifications to the Session Manager (and no other existing components) are required to add a new type of client. Each client would load a data type table into the Database.

We are attempting to generalize the abstract implementation architecture to be configurable for any learning application that requires networked collaboration, coaching, and multimedia. Adaptive multimedia [1] could be included with scripts that automatically generate HTML pages from the database to meet user's needs. We have designed and implemented a prototype of this adaptive hypermedia extension but have not incorporated into our released system. Student modeling facilities would be improved by informing the Coach of which materials students have examined via the Tracker.

6. Resource Layer

At this level one describes the system in terms of the resources used and their performance characteristics, including performance of both hardware and implemented software, as well as constraints on where that software resides. In Belz and Luckham's work this level of description is used primarily for performance modeling,[4] which is not a concern in this paper. For present purposes the most significant resource constraints on the implemented architecture are as follows:

Client platforms: Any platform supporting Java applications and Netscape 2.0 or better. We have tested on Mac OS, Solaris, Windows'95 and Windows NT, and are working on Windows 3.1. Current installations in our lab and in DoDDS schools are on PowerMacintosh 8100 series and various

[4] See http://juicy.dh.trw.com:8090/tutorial/index.html for a tutorial on modeling in Rapide, using the architecture described in this paper as an example.

Pentium platforms. The applications shown in the shaded box in Figure 3 must be running at the same IP location.

Server platforms: Currently a Unix server is required. The redesigned version will deliver on Windows NT and other server class machines. The server is currently installed on a Sparcstation 20 MP in our lab and on Netras in each of the 4 DoDDS schools. The server components shown in shaded boxes must reside at the same location as others in the box.

Network requirements: With the possible exception of images embedded in HTML materials, all messages are small, so communication load is low. A 28.8 connection is adequate. Current installations are all 10BaseT.

7. Conclusions

The advent of the Web has brought us widespread connectivity, shared protocols, and software languages that can migrate between platforms. These have enabled the development of client-server systems for delivery of interesting functionality as well as materials, on a variety of platforms. Such systems provide the AI&ED community with more viable options for getting systems delivered in the "real world." During development we can choose to use sophisticated tools for knowledge-based systems, and to the extent that connectivity is available, deliver intelligent functionality without needing to scale down the intelligence. Furthermore, this new technology can help us address some of the pragmatic problems that have plagued the AI&ED community and others who are developing knowledge-based applications. We have begun to resolve some of the basic interoperability issues that will make it easier to reuse components of ITS and other knowledge-based systems. This reuse will enable researchers to allocate more effort to research rather than development of the infrastructure needed to test their ideas, as well as reducing cost of delivery. The real issues -- the hard ones -- are now shifting to a more conceptual level of analysis. We need to address the issue of how we can share *content*, including media, pedagogical strategies, and intelligent services such as user modeling. Shared ontologies may be a step in this direction.

8. Acknowledgments

We express gratitude to Kim Harrigal for work on the Client, Joe Toth for work on the Coach, and Frank Belz for interesting discussions about architecture. Project members Alan Lesgold (PI), Sandy Katz and Arlene Weiner (co-PIs with Suthers), and Eva Toth (postdoc) contributed to the interface design and curriculum materials. Funded by ARPAs Computer Aided Education and Training Initiative, under the title "Collaboration, Apprenticeship, and Critical Discussion: Groupware for Learning", Contract N66001-95-C-8621.

9. References

[1] Brusilovsky, P., Schwarz, E, & Weber, G. (1996). ELM-ART: An intelligent tutoring system on world wide web. ITS'96, Third International Conference on Intelligent Tutoring Systems Monteal, June 1996, pp. 261-269.

[2] Ikeda, M., Hoppe, U., & Mizoguchi, R. (1995). Ontological Issues of CSCL Systems Design. AI-Ed 95, the 7th World Conference on Artificial Intelligence in Education., August 16-19, 1995, Washington DC, pp.242-249.

[3] Murray, T. (1996) Having it all, maybe: Design tradeoffs in ITS authoring tools. ITS'96, Third International Conference on Intelligent Tutoring Systems Monteal, June 1996, pp. 93-101.

[4] O'Neill, D. K., & Gomez, L. M. (1994).The collaboratory notebook: A distributed knowledge-building environment for project-enhanced learning. In Proceedings of Ed-Media '94, Vancouver, BC.

[5] Paolucci, M., Suthers, D. and Weiner, M. (1995). Belvedere: Stimulating Students' Critical Discussion. CHI95 Conference Companion, May 7-11 1995, Denver CO, pp. 123-124.

[6] Ritter, S. and Koedinger, K. (1995). Towards lightweight tutoring agents. AI-Ed 95, the 7th World Conference on Artificial Intelligence in Education., August 16-19, 1995, Washington DC, pp. 91-98.

[7] Roschelle, J. & Kaput, J. (1995). Educational software architecture and systemic impact: The promise of component software. Presented at AERA Annual Meeting, San Francisco, April 19, 1995.

[8] Scardamalia, M., & Bereiter, C. (1991). Higher levels of agency for children in knowledge building: A challenge for the design of new knowledge media.The Journal of the Learning Sciences, 1(1), 37--68.

[9] Slavin, R. E. (1990).Cooperative learning: Theory, research, and practice. Englewood Cliffs, NJ: Prentice-Hall.

[10] Smolensky, P., Fox, B., King, R., & Lewis, C. (1987). Computer-aided reasoned discourse, or, how to argue with a computer. In R. Guindon (Ed.), Cognitive science and its applications for human-computer interaction (pp. 109-162). Hillsdale, NJ: Erlbaum.

[11] Streitz, N. A., Hannemann, J., & Thuring, M. (1989). From ideas and arguments to hyperdocuments: Traveling through activity spaces. In Hypertext '89 Proceedings, Pittsburgh, PA (pp. 343--364). New York: ACM.

[12] Suthers, D., Weiner, A., Connelly, A. and Paolucci, M. (1995). Belvedere: Engaging students in critical discussion of science and public policy issues.AI-Ed 95, the 7th World Conference on Artificial Intelligence in Education., August 16-19, 1995, Washington DC

[13] Suthers, D. and Weiner, A. (1995). Groupware for developing critical discussion skills. CSCL '95, Computer Supported Cooperative Learning, Bloomington, Indiana, October 17-20, 1995.

[14] Toth, J., Suthers, D., and Weiner, A. (1997). Providing expert advice in the domain of collaborative scientific inquiry. To appear in AI&ED97.