THE AUTOTUTOR 3 ARCHITECTURE
A software architecture for an expandable, high-availability ITS

Patrick Chipman, Andrew Olney, Arthur C. Graesser
Institute for Intelligent Systems, University of Memphis, 365 Innovation Drive, Memphis, TN, USA

Keywords: Intelligent tutoring systems, software architecture, Internet-based instruction

Abstract: Providing high quality of service over the Internet to a variety of clients while simultaneously providing good pedagogy and extensibility for content creators and developers are key issues in the design of the computational architecture of an intelligent tutoring system (ITS). In this paper, we describe an ITS architecture that attempts to address both issues using a distributed hub-and-spoke metaphor similar to that of the DARPA Galaxy Communicator. This architecture is described in the context of the natural language ITS that uses it, AutoTutor 3.

1 INTRODUCTION

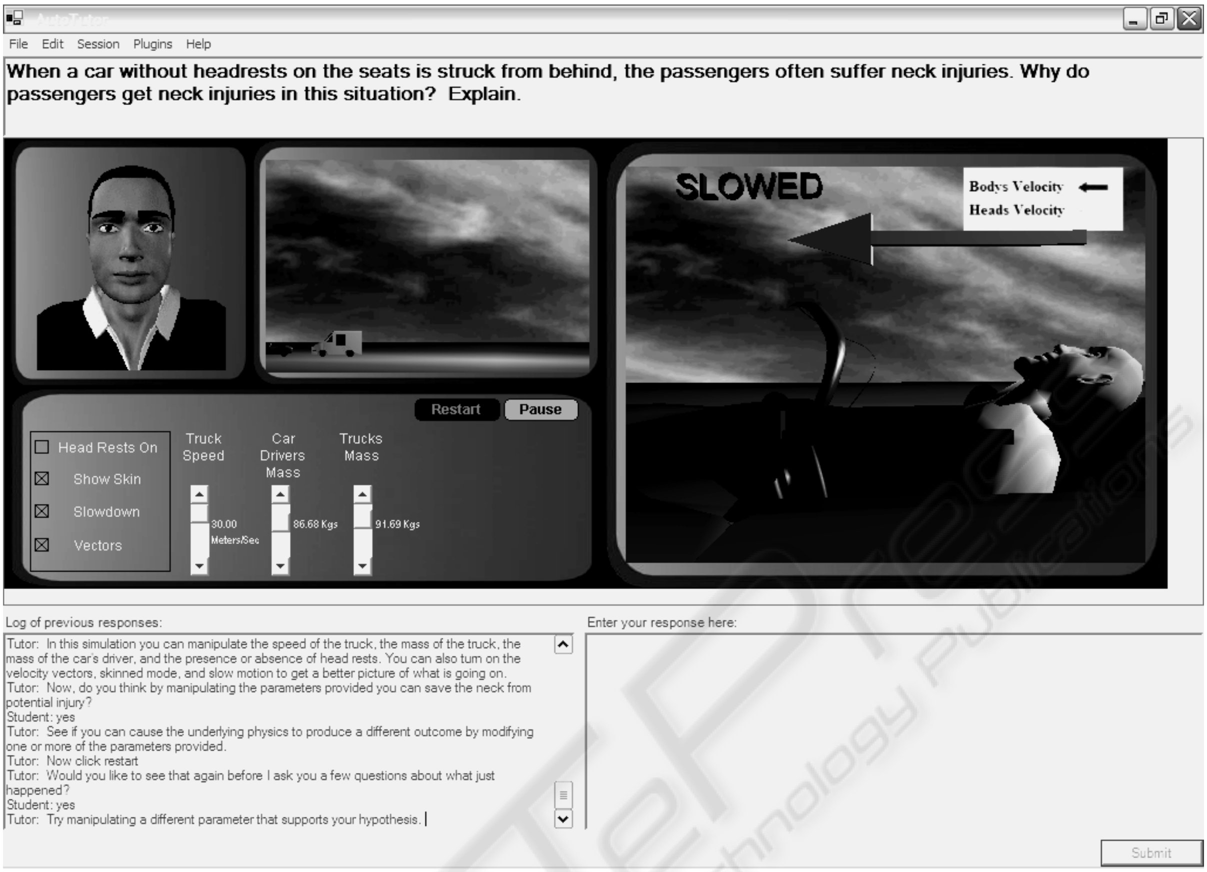
A great deal of advancement in the state of the art of intelligent tutoring systems (ITS) has occurred in the last several years. Primarily, these advancements have been focused on improving pedagogical strategies by incorporating established psychological research on human tutoring into tutoring systems (Graesser, Person, & Magliano, 1995; Aleven & Koedinger, 2002; VanLehn, Jones, & Chi, 1992); adding superior student knowledge modelling such as model-tracing (VanLehn et al., 2000); providing advanced authoring tools to facilitate the rapid use of the ITS in new domains of knowledge, or with different sets of learners with different levels of skills (Ainsworth & Grimshaw, 2002); or improving the interface by adding animated characters such as "talking heads," also known as animated pedagogical agents (Johnson, Rickel, & Lester, 2000), or natural language dialogue (Jordan, Rosé, & VanLehn, 2001).

However, behind all of these systems and their advancements must reside some form of computational architecture. In many cases, this architecture is monolithic, rarely discussed, and generally irrelevant. Intelligent tutoring systems that reside on modern desktop computers have vast resources available for their processing and a high user tolerance for failure, especially if the system is visually appealing, quick to respond, and otherwise meets the user's typical expectations of a "typical" application (Bouch & Sasse, 1999). In fact, the presence of an animated pedagogical agent can improve the subjective likeability of a system considerably (Moreno, Klettke, Nibbaragandla, Graesser, & TRG, 2002), which would further enhance the user's experience and allow him to overlook any flaws in the underlying software (Bouch & Sasse, 1999).

However, for a web-based or Internet-based system, where the target platform's resources are often much lower than that of a modern desktop computer and much of the processing must be handled on a remote server for potentially hundreds or thousands of simultaneous users, architectures that provide consistent levels of availability and latency are mandatory if the system is to be adopted by users (Bhatti, Bouch, & Kuchinsky, 2000). Furthermore, such architectures must be able to handle the sorts of advancements in ITS technology that come at a rapid pace while simultaneously allowing developers and content creators to achieve domain and tutoring strategy independence. If all of these criteria are not met to some degree, it is probable that user acceptance, both with learners and content creators, will be low and will confine the ITS to laboratory use.

In this paper, we discuss the architecture of the third version of the venerable AutoTutor natural dialogue intelligent tutoring system. This architecture was designed specifically to balance the criteria of high availability and expandability, thereby offering a quality user experience while providing the extensibility necessary for the creation of more advanced ITSes in the future. Additionally, the architecture is sufficiently generic that other

Chipman P., Olney A. and C. Graesser A. (2005).
THE AUTOTUTOR 3 ARCHITECTURE - A software architecture for an expandable, high-availability ITS.
In *Proceedings of the First International Conference on Web Information Systems and Technologies*, pages 466-473
DOI: 10.5220/0001233504660473



When a car without headrests on the seats is struck from behind, the passengers often suffer neck injuries. Why do passengers get neck injuries in this situation? Explain.

Log of previous responses:

Tutor: In this simulation you can manipulate the speed of the truck, the mass of the truck, the mass of the car's driver, and the presence or absence of head rests. You can also tum on the velocity vectors, skinned mode, and slow motion to get a better picture of what is going on.
Tutor: Now, do you think by manipulating the parameters provided you can save the neck from potential injury?
Student: yes
Tutor: See if you can cause the underlying physics to produce a different outcome by modifying one or more of the parameters provided.
Tutor: Now click restart
Tutor: Would you like to see that again before I ask you a few questions about what just happened?
Student: yes
Tutor: Try manipulating a different parameter that supports your hypothesis.

Enter your response here:

Figure 1: The AutoTutor 3 user interface.

systems can be built around its principles; it is not solely restricted to use with our AutoTutor system.

2 WHAT IS AUTOTUTOR?

A discussion of the architecture of AutoTutor 3 would not be complete without an explanation of the system itself. AutoTutor is a complex system that simulates a human or ideal tutor by holding a conversation with the learner in natural language (Graesser, Lu, et. al., in press). AutoTutor presents a series of questions or problems that require approximately a paragraph of information to answer correctly. An example question in conceptual physics is “When a car without headrests on the seats is struck from behind, the passengers often suffer neck injuries. Why do passengers get neck injuries in this situation?” A complete answer to this question is approximately 3-7 sentences in length. AutoTutor assists the learner in the construction of an improved answer that draws out more of the learner’s knowledge and that adaptively corrects problems with the answer. The dialogue between AutoTutor and the learner typically lasts 50-200 conversational turns for one question. Figure 1 shows an example of the AutoTutor 3 interface.

The AutoTutor system has undergone a variety of empirical tests to validate its pedagogical and conversational efficacy in both the domains of computer literacy (Graesser, Lu, et. al., in press) and conceptual physics (Graesser, Jackson, et. al., 2003). A “bystander Turing test” was performed to validate AutoTutor’s conversational smoothness. In such an experiment, a subject is shown a section of tutorial dialogue randomly selected from real AutoTutor transcripts in which, half the time, the tutor move generated by AutoTutor has been replaced by a move generated by a human expert tutor. The subjects in this experiment, the bystanders, are asked to specify if the tutor move in question was generated by a human or a computer. The bystanders were wholly unable to make this distinction (Bautista, Person, & Graesser, 2002). Tests of pedagogical effectiveness have shown learning gains of 0.2 to 1.5 sigma (standard deviation units) with a

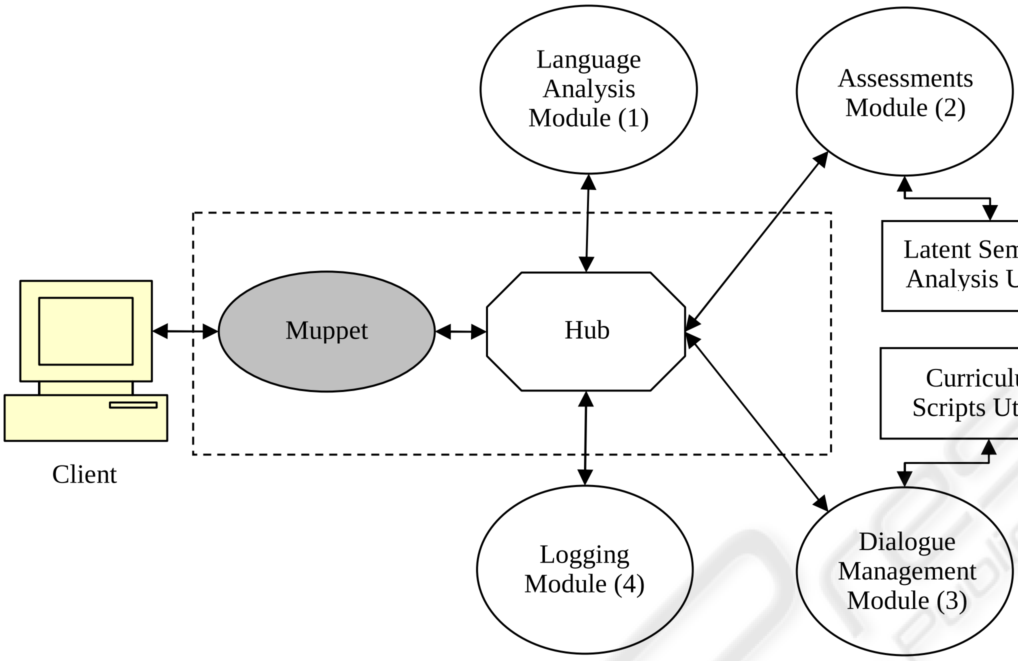

Language Analysis Module (1)
Assessments Module (2)
Latent Semantic Analysis Utility
Muppet
Hub
Client
Curriculum Scripts Utility
Logging Module (4)
Dialogue Management Module (3)

Figure 2: An overview of the AutoTutor 3 Architecture.

mean of 0.8 sigma or around one letter grade of improvement. The performance varies based on the type of measure used and the content domain (Graesser, Jackson, et al., 2003). This is comparable to both the performance of unskilled human tutors, who produce learning gains of around 0.4 sigma, or half a letter grade of improvement (Cohen, Kulik, & Kulik, 1982), as well as to the performance of other intelligent tutoring systems without natural language dialogue, which produce learning gains of around 1.0 sigma (Corbett, 2001).

3 SYSTEM ARCHITECTURE

It has long been the desire of the Tutoring Research Group to offer AutoTutor to the widest audience possible, both in terms of learners and content creators, because of its impressive performance in empirical testing. As opposed to many intelligent tutoring systems, AutoTutor offers a natural language interface; this is posited to be critical for future ITS development (Jordan, Rosé, & VanLehn, 2001). However, this natural language interface requires a great deal of computational resources in both processing power and storage, making it difficult to deploy to desktop computers that are not state of the art. Furthermore, content creators offer up a great deal of intellectual property when creating the curriculum scripts that dictate the output of the system. It is unlikely these individuals will be willing to provide their content for local use by any number of learners. To solve both of these problems, it was decided to utilize a client-server architecture in which the AutoTutor 3 server resides at a fixed location and learners and content creators access its functionality remotely.

The system architecture is somewhat related to the DARPA Galaxy Communicator model, in which a variety of modules communicate, mediated by a central "hub" (Galaxy Communicator Documentation). In the AutoTutor 3 architecture, outlined in Figure 2, a central object known as the Hub (the octagon), hosted in the AutoTutor 3 server software, passes an object that contains the state of the system, the State Table (not shown), between a set of Modules (circles) that alter the state without having any specific knowledge of each other; the order of this process is specified by the Hub and, for the current AutoTutor 3 system, is expressed in the figure as a number after each Module's name. Each Module may access a number of Utilities (squares) that provide services through published interfaces. The State Table is sent to a variety of potential client types using one of many Multi-Protocol Personal Translators, or Muppets (shaded circle), that convert the State Table into a format that the client can understand. The objects contained within the dashed rectangle exist together in the main AutoTutor 3 server; all of the other objects are served through our custom-written generic object server, the Module Server, and can each exist on the same or different machines as load demands.

3.1 The .NET Framework and Remoting

The AutoTutor 3 system and its underlying architecture are implemented in a combination of C# and Visual Basic .NET, using the .NET Framework version 1.1 and the Common Language Runtime by Microsoft Corporation. The CLR provides a variety of advantages, not the least of which is the generic remote procedure call system known as .NET Remoting. This part of the Framework allows remote objects to be accessed as if they were inside the AutoTutor server process; short of a call into the Framework to “activate” the target object (whether it is a Module or a Utility), the object can be accessed identically across the network or on the local machine (Microsoft .NET Technology Overview). By using the Remoting system, it is possible for AutoTutor 3 Modules and Utilities to be split across multiple computers or multiple processes as required to “scale out” as load increases. The underlying complexities of accessing these remote objects are hidden behind the AT3Communicator class and the Remoting system.

Remoting provides a binary communication channel that, in our internal tests, allows the entire State Table for any turn to be conveyed using under 12 kilobytes of data, thereby reducing network transfer latency within the system and to clients. Our testing of the server under common experimental loads of around 30 simultaneous users reveals that the network latency of a system where the Modules and Utilities exist on separate machines is less than 1 ms, given a 100BaseT Ethernet interconnect.

3.2 State Table

In many ways, the State Table is the core of the AutoTutor 3 architecture. This extensible class contains the complete state of the system for any particular student interaction with the tutor. It normally survives for an entire problem and is discarded at the end of a problem. The State Table provides a logical separation of the data upon which the Modules work from the algorithms of the Modules themselves; in this way, it acts both as the storage space for the system’s student model, as well as a sort of command object if one considers the architecture as an implementation of the chain-of-responsibility design pattern. Individual Modules store the results of their processing in the State Table. These results can then be read and further processed by other Modules, or simply ignored by other Modules if they are irrelevant to their processing. Because the state of the system is loosely coupled to the Modules that use it, it is relatively easy for new Modules to be added to the system to work on the data contained within the State Table.

The State Table is a class that is tied to a specific inheritance chain of interfaces. This ensures that Modules are themselves loosely coupled to the internal structure of the State Table; a Module created for an earlier implementation and older interface is guaranteed to work with newer versions of the State Table, because backward compatibility is mandated by the interface.

3.3 Hub

The Hub is the central manager of the AutoTutor architecture. This extremely simple class has only one function: to call each of the Modules of which it knows in the sequence required to produce a complete State Table. The AutoTutor server software handles loading the Module references into the Hub, which then makes the calls using Remoting. While it would seem necessary to rewrite the Hub whenever adding a new Module to the system, the current implementation of the Hub calls each of the Modules of which it knows in the sequence in which they were loaded; as this load sequence can be specified to the server in its configuration file, as long as dependencies in which Modules must be called more than once are avoided, the standard Hub implementation should be sufficient.

3.4 Modules and the AT3Communicator

Each Module in the system, as shown in Figure 2, represents a separate stage in the processing of a student move and the generation of an appropriate tutor turn. The internal mechanisms AutoTutor uses in each of those stages are covered in detail elsewhere (Olney, Louwerse, Mathews, Marineau, Mitchell, & Graesser, 2003; Mathews, Jackson, Olney, Chipman, & Graesser 2003; Graesser, Lu, et al. in press) and will not be detailed here. Each Module inherits from a master class called “AT3Communicator,” which encapsulates the necessary public methods and implementations to link the Module to the system by taking messages and their associated State Tables from the Hub, acquiring references to the Utility objects, and handling thread synchronization should the Module be called by multiple users, and therefore multiple threads of processing, at once.

Because all of this functionality is encapsulated in this base class, those who wish to extend the capabilities of the AutoTutor system by adding a new Module or altering an existing one need only override a single virtual method called “Execute,” which is analogous to a “Main” function in standard procedural programming. This overridden method is called by the base class and a copy of the State Table is passed in; the Module returns this copy with any necessary modifications. Utilities may be called by reading their references from a hash table, then calling methods on those references. The Remoting system, as previously mentioned, handles the resolution of those method calls.

While the AT3Communicator base class does handle thread synchronization with regards to the State Table itself and the Utility references, thread safety is not assured if the Module developer opts to add member variables to his Module’s class. However, this problem can be readily avoided by using static variables in the Execute method and following standard programming practices that argue against the use of global variables; alternatively, the Module programmer can use the State Table to store the internal state of his module between calls. The current Dialogue Management Module uses this technique.

3.5 Utilities

The Utilities of the architecture are external objects called by Modules using Remoting. Unlike Modules, these objects have no fixed base class or interfaces, nor are they called by the Hub. Therefore, thread safety is not hidden from the developer. The complexities of Remoting are hidden from the Utility developer by the Module Server, however. In return, the developer of a Utility receives the flexibility to define his own interface and further gains the ability for his object’s methods to be called directly from Modules, which can then share its functionality. In AutoTutor 3, we have chosen to use Utilities to encapsulate functionality used by multiple Modules, such as the Latent Semantic Analysis used to evaluate the similarity of strings (and thus the quality of student responses), or the Curriculum Scripts that dictate the pedagogical moves of the system and provide domain independence, as detailed by Mathews et al. (2003).

3.6 Muppets

Multi-Protocol Personal Translators (“Muppets”) are the “glue” that connect clients to the system. They exist within the main AutoTutor server and translate the State Table into a format that a client can understand. Muppets allow the server to connect to clients in any programming language with any set of capabilities; smart clients written in a .NET language can connect to a Remoting Muppet, for instance, and have access to the entire State Table. A web browser could connect to a Web Server Muppet that turns the State Table into a web page with sufficient session management to keep track of each user connecting to the web site. Mobile phones could use an Instant Messaging Muppet that emulates an Instant Messaging service or chat room.

Muppets are perhaps the most complicated part of the system to develop, as they must deal with session management and network protocols; none of these low level details are hidden. To facilitate Muppet development and use of AutoTutor on multiple platforms, the architecture was developed with three Muppets: a .NET Remoting Muppet for smart clients, a text-based Muppet that uses simple TCP sockets, and a web-based Muppet that provides a simple World Wide Web interface.

3.7 Server Software

The AutoTutor 3 architecture uses only two pieces of server software: the AutoTutor Server, which handles Muppets and Hubs, and the Module Server, which is a generic server for offering .NET objects over Remoting. The AutoTutor Server is designed to bootstrap the entire system by using its configuration file to locate, instantiate, and initialize Muppets, Hubs, and all of the Modules and Utilities used by them. Each instance of an AutoTutor Server is capable of handling multiple Muppets and Hubs with the same or different sets of Modules and Utilities, which gives it the ability to support different “versions” of AutoTutor on a single machine that differ only in their interface to clients or in their internal processing steps.

The Module Server is not specific to this architecture. It is simply a generic server that can instantiate and offer objects or parts of objects, as defined by interfaces, through .NET Remoting. It is crucial to the proper operation of the architecture, but it can be used by any project in which Common Language Runtime objects need to be offered. Other distributed systems may readily make use of this server without implementing any part of the AutoTutor 3 architecture.

3.8 Client Software

Through the use of Muppets, specific client software is not required to use systems built on the AutoTutor 3 architecture. However, a smart client with support for plug-ins, an animated pedagogical agent, 3-D simulations, and client-side processing of data is available. Additionally, the Web Muppet provides a text-based interface on the World Wide Web.

4 EMPIRICAL TESTS OF PERFORMANCE

The AutoTutor 3 system was completed approximately one year ago. It is a complete rewrite of the older AutoTutor 2 system (Graesser, VanLehn, Rosé, Jordan, & Harter, 2001). As such, empirical tests both of its ability to mimic this older system's abilities while adding new functionality and also of its architecture's raw performance are ongoing. Thus far, empirical tests look promising, with the AutoTutor 3 system matching the pedagogical performance of the AutoTutor 2 system and further enhancing it with the addition of 3-D simulations within the domain of conceptual physics.

With regards to the architecture's performance, internal profiling reveals that network latencies between components are less than 1 ms, though this is of course likely to increase if the components are further separated over a larger network. The Modules and Utilities of AutoTutor are CPU bound; their memory requirements are roughly constant, requiring only approximately another 100 kilobytes per simultaneous user atop a basic memory footprint of approximately 180 megabytes. Again, these values will vary based on the Modules used, but profiling shows that the architecture itself contributes very little to the memory or CPU footprint of the AutoTutor 3 processes.

Based on the average size of the State Tables in our internal stress testing using active users and distributed load generation with multiple computers, we estimate that any individual AutoTutor server instance can support at maximum approximately 800 simultaneous users, assuming all of the components of the system are located on a single server machine (a Pentium Xeon 1.4 gigahertz with 1 gigabyte of RAM in our tests) and the clients connect using 100BaseT Ethernet. Our testing of the system's architectural performance in the course of empirical testing of its pedagogy shows that it can readily support at least 30 simultaneous users with no detectable loss of responsiveness. A large study in which the system is used to support remote, naïve learners at other universities is in progress, but preliminary results have shown that a single AutoTutor 3 server is more than capable of providing advanced, natural language intelligent tutoring services to several hundred simultaneous users across the Internet while maintaining a high quality of service. Further empirical testing of the system's performance in the context of new experiments is currently in progress and should be completed by the end of 2005.

5 FUTURE DIRECTIONS

Beyond the need for further empirical performance testing, there is room for improvement in this architecture. At the moment, any form of load balancing or clustering must be handled manually by those hosting AutoTutor servers; monitoring application load and responding to it is a difficult and time-consuming task for system administrators. Future versions of this architecture, which will maintain backwards compatibility and provide these advantages to all existing code by leveraging the class inheritance system, will provide adaptive load balancing services through the strategic use of threading and dynamic load shedding. This will allow other computers to dynamically take over parts of the AutoTutor processing when the server is overloaded, or will allow a Muppet to transparently redirect a learner to a less crowded server providing the same content. Techniques such as the independent event queues and controllers of the SEDA architecture (Welsh, Culler, & Brewer, 2001) may be used to provide better quality of service under extremely heavy loads.

To make the AutoTutor system itself and not just its architecture more appealing to content creators, support for authoring tools that can manipulate the internal state of the Modules (such as the pedagogical strategies of the Dialogue Management Module) will be added, along with licensing support that can restrict use of tutoring systems based on this architecture, including AutoTutor, to those authorized to use the intellectual property contained within.

AUTHOR NOTE

The Tutoring Research Group (TRG) is an interdisciplinary research team comprised of approximately 35 researchers from psychology, computer science, physics, and education (visit

http://www.autotutor.org). The research on AutoTutor was supported by the National Science Foundation (SBR 9720314, REC 0106965, REC 0126265, ITR 0325428) and the DoD Multidisciplinary University Research Initiative (MURI) administered by ONR under grant N00014-00-1-0600. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DoD, ONR, or NSF.

REFERENCES

Ainsworth, S. & Grimshaw, S (2002). Evaluating the Effectiveness and Efficiency of the REDEEM Intelligent Tutoring System Authoring Tool. Retrieved October 26, 2004, from the University of Nottingham, ESRC Centre for Research in Development, Instruction, and Training web site: *www.psychology.nottingham.ac.uk/staff/sea/techreport_69.pdf*

Aleven, V. & Koedinger, K. R. (2002). An effective metacognitive strategy: Learning by doing and explaining with a computer-based Cognitive Tutor. *Cognitive Science*, *26*, 147-179.

Bautista, K. Person, N., Graesser, A. C., & Tutoring Research Group (2002). Human or computer? AutoTutor in a bystander Turing test. In *Proceedings of the Sixth International Conference on Intelligent Tutoring Systems* (pp. 821-830). Berlin, Germany: Springer-Verlag.

Bhatti, N., Bouch, A., & Kuchinsky, A. (2000). Integrating User-Perceived Quality into Web Server Design. In Proceedings of the Ninth International World Wide Web Conference. May 2000. Amsterdam. Retrieved January 22, 2005 from: http://www9.org/w9cdrom/92/92.html

Bouch, A. & Sasse, M. A. (1999). It ain't what you charge it's the way that you do it: A user perspective of network QoS and pricing. In *Proceedings of IM'99*. Boston: IFIP.

Cohen, P. A., Kulik, J. A., & Kulik, C. C. (1982). Educational outcomes of tutoring: A meta-analysis of findings, *American Educational Research Journal*, *19*, 237-248.

Corbett, A. T. (2001). Cognitive computer tutors: Solving the two-sigma problem. In *User Modeling: Proceedings of the Eighth Annual Conference* (pp. 137-147). Berlin, Germany: Springer-Verlag.

Galaxy Communicator Documentation. (n.d.) Retrieved December 17, 2004, from http://communicator.sourceforge.net/sites/MITRE/distributions/GalaxyCommunicator/docs/manual/index.html

Graesser, A. C., Jackson, G. T., Mathews, E. C., Mitchell, H. H., Olney, A., Ventura, M., et al. (2003). Why/AutoTutor: A test of learning gains from a physics tutor with natural language dialogue. In *Proceedings of the 25th Annual Conference of the Cognitive Science Society* (pp. 1-6). Boston, MA: Cognitive Science Society.

Graesser, A. C., Lu, S., Jackson, G. T., Mitchell, H., Ventura, M., Olney, A., & Louwerse, M. M. (in press). AutoTutor: A tutor with dialogue in natural language. *Behavioral Research Methods, Instruments, and Computers*.

Graesser, A. C., Person, N. K., & Magliano, J. P. (1995). Collaborative dialogue patterns in naturalistic one-on-one tutoring. *Applied Cognitive Psychology*, *9*, 359-387.

Graesser, A. C., VanLehn, K., Rosé, C., Jordan, P., & Harter, D. (2001). Intelligent tutoring systems with conversational dialogue. *AI Magazine*, *22*, 39-51.

Johnson, W. L., Rickel, J. W., & Lester, J. C. (2000). Animated pedagogical agents: Face-to-face interaction in interactive learning environments. *International Journal of Artificial Intelligence in Education*, *11*, 47-78.

Jordan, P., Rosé, C., & VanLehn, K. (2001). Tools for authoring tutorial dialogue knowledge. In *Proceedings of AI in Education 2001 Conference*. May 2001. Amsterdam: IOS Press.

Mathews, E. C., Jackson, G. T., Olney, A., Chipman, P. & Graesser, A. C. (2003). Achieving Domain Independence in AutoTutor. In N. Callaos, M. Margenstern, J. Zhang, O. Castillo, E. Doberkat (Eds.), *The Seventh World Multiconference on Systemics, Cybernetics, and Informatics Proceedings: Computer Science and Engineerings I, Vol. 5* (pp. 172-176). Orlando: IIIS.

Microsoft .NET Technology Overview. (n.d.) Seattle, WA: Microsoft Corp. Retrieved October 11, 2004, from: http://msdn.microsoft.com/netframework/technologyinfo/overview/default.aspx

Moreno, K. N., Klettke, B., Nibbaragandla, K., Graesser, A. C., & the Tutoring Research Group (2002). Perceived characteristics and pedagogical efficacy of animated conversational agents. In S. A. Cerri, G. Gouarderes, & F. Paraguacu (Eds.), *Intelligent Tutoring Systems 2002* (pp. 963-971). Berlin, Germany: Springer.

Olney, A., Louwerse, M. M., Mathews, E. C., Marineau, J., Mitchell, H. H., & Graesser, A. C. (2003). Utterance classification in AutoTutor. In *Building Educational Applications using Natural Language Processing: Proceedings of the Human Language Technology - North American Chapter of the Association for Computational Linguistics Conference 2003 Workshop* (pp. 1-8). Philadelphia: Association for Computational Linguistics.

VanLehn, K., Freedman, R., Jordan, P., Murray, C., Osan, R., Ringenberg, M., Rosé, C., Schulze, K., Shelby, R., Treacy, D., Weinstein, A., & Wintersgill, M. (2000). Fading and Deepening: The Next Steps for Andes and Other Model-Tracing Tutors. In *Intelligent Tutoring Systems: Fifth International Conference (ITS 2000)*, Montreal. Springer-Verlag Lecture Notes in Computer Science.

VanLehn, K., Jones, R. M., & Chi, M. T. H. (1992). A model of the self-explanation effect. *Journal of the Learning Sciences*, *2*, 1-60.

Welsh, M., Culler, D., & Brewer, E. (2001). SEDA: An architecture for well-conditioned, scalable internet services. In *Proceedings of the Eighteenth Symposium on Operating Systems Principles (SOSP-18)* (pp. 230-243). ACM Press.