The eXtensible Tutor Architecture: A New Foundation for ITS

Goss NUZZO-JONES, Jason A. WALONOSKI, Neil T. HEFFERNAN, Tom LIVAK
Worcester Polytechnic Institute
100 Institute Rd, Worcester, MA 01609
(508) 831-5569
goss@wpi.edu, jwalon@wpi.edu, nth@wpi.edu, tomlivak@alum.wpi.edu

Abstract. The eXtensible Tutor Architecture (XTA) was designed as a platform for creating and deploying many types of Intelligent Tutoring Systems across many different platforms. The XTA presently has support for state graph pseudo-tutors and JESS model-tracing cognitive tutors, in both a client and server context. Supported interfaces are presently Java Swing / WebStart and HTML. The XTA was designed with future development in mind, allowing easy specification of new tutor types, tutoring strategies, and interface layers. It has been used as the foundation of the Assistments Project, a wide scale server based ITS deployment. The Assistments Project is on track to provide ITS content to 100,000 students in the state of Massachusetts.

1. Introduction & Background

This research was conducted to develop a scalable, stable framework for deploying Intelligent Tutoring Systems (ITS) of many types to a variety of platforms. The term Intelligent Tutoring Systems covers a wide range of possible computer-based tutors, from cognitive model tracing tutors [3], constraint-based tutors [10], to pseudo-tutors. Pseudo-tutors are simplified cognitive models based on state graphs. The state graphs of pseudo-tutors are finite graphs with each node representing a state of the interface, and each arc representing an action made by a student. Student actions trigger transitions in the graph, and the current state of the problem is represented by the graph. Pseudo-tutors are behaviorally equivalent to rule-based tutors [1]. Our research attempted to support all these types of tutors, but provide a clear path for future development and customization.

Additionally, our research was dependent on the needs of the Assistments Project. This project required that we be able to support the full range of tutors, provide stability and scalability, and deliver tutoring content to a host of clients – either rich client applications such as Java WebStart, or thin light-weight HTML clients (possibly enriched by scripts and Macromedia Flash™). To accomplish these client interface goals, we were required to follow software engineering practice by cleanly separating the logic and presentation of tutors.

The success of ITS in general is well known, demonstrating useful learning effects [8]. There have been ITS that have been deployed on a wide scale [8], but they suffered from some limitations, such as a lack of centralized logging, upgrade difficulties, and tutor strategy inflexibility. It has been shown that centralized logging of student actions in databases for experimental analysis is valuable [11]. Our research sought to address these issues, as well as provide a rich feature base for future development of all tutor types.

Other projects [7] have sought to provide a rapid development environment and stable runtime for deploying individual tutoring applications. However, this approach also has shortcomings in terms of wide deployment and scalability, as well as separation of logic and presentation. We attempted to resolve these problems by creating an environment that can support many tutoring strategies (including those mentioned above), operate as both a client and scaleable server application, provide logging capabilities for student analysis, and remain highly extensible for future development. The results of this research were used as the deployment mechanism for the Assistments Project, a mathematics ITS project based at Worcester Polytechnic Institute and Carnegie Mellon University [12].

1.1 The eXtensible Tutor Architecture

The result of our research is a framework that we refer to as the eXtensible Tutor Architecture (XTA). This framework controls the interface and behaviors of our intelligent tutoring system via a collection of modular units. These units conceptually consist of a *curriculum* unit, a *problem* unit, a *strategy* unit, and a *logging* unit. Each conceptual unit has an abstract and extensible implementation allowing for evolving tutor types and content delivery methods.

The XTA is represented by the diagram given in Figure 1, illustrating the actual composition of the units. This diagram shows the relationships between the different units and their hierarchy. Within each unit, the XTA has been designed to be highly flexible in anticipation of future tutoring methods and interface layers. This was accomplished through encapsulation, abstraction, and clearly defined responsibilities for each component. These software engineering practices allowed us to present a clear developmental path for future components. That being said, the current implementation has full functionality in a variety of useful contexts.

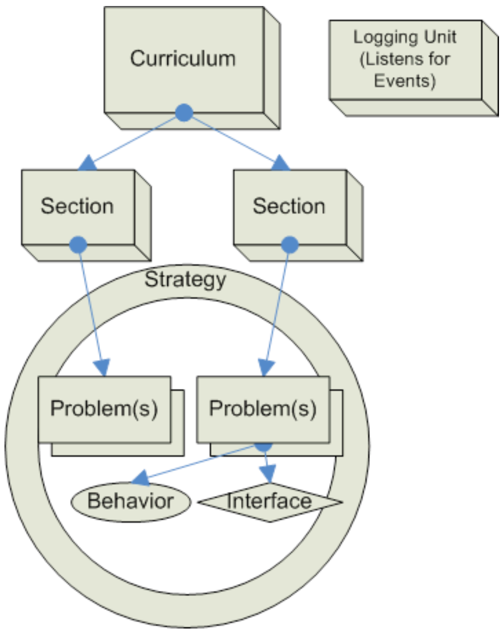


Figure 1 - Abstract Unit Diagram

1.1.1 Curriculum Unit

The *curriculum* unit can be conceptually subdivided into two main pieces: the *curriculum* itself, and *sections*. The *curriculum* is composed of one or more *sections*, with each *section* containing *problems* or other *sections*. This recursive structure allows for a rich hierarchy of different types of *sections* and *problems*.

Progress within a particular *curriculum*, and the *sections* of which is it composed, are stored in a *progress file* – an XML meta-data store that indexes into the *curriculum* and the current *problem* (one progress file per student per curriculum).

The *section* component is an abstraction for a particular listing of problems. This abstraction has been extended to implement our current *section* types, and allows for future expansion of the *curriculum* unit. Currently existing *section* types include "Linear" (*problems* or sub-*sections* are presented in linear order), "Random" (*problems* or sub-*sections* are presented in a pseudo-random order), and "Experiment" (a single *problem* or sub-*section* is selected pseudo-randomly from a list, the others are ignored). Plans for future *sections* types include a "Directed" *section*, where *problem* selection is directed by the student's knowledge model [2].

1.1.2 Problem Unit

The *problem* unit represents a problem to be tutored, including questions, answers, and relevant knowledge-components required to solve the problem. For instance pseudo-tutors are a hierarchy of questions connected by correct and incorrect answers, along with hint messages and other feedback. Each of these questions is represented by a *problem* composed of two main pieces: an *interface* and a *behavior*.

The *interface* definition is interpreted by the runtime and displayed for viewing and interaction to the user. This display follows a two-step process, allowing for easy customization to platform and interface specifications. The *interface* definition consists of "high-level" *interface* elements ("widgets"), which can have complex behavior (multimedia, spell-checking text fields, algebra parsing text fields). These "high-level" widgets have a representation in the runtime composed of "low-level" widgets. "Low-level" widgets are widgets common to many possible platforms of *interface*, and include text labels, text fields, images, radio buttons, etc. These "low-level" widgets are then consumed by an *interface* display application. Such applications consume "low-level" widget XML, and produce an interface on a specific platform. At present we have implemented a Java Swing *interface* display application, and a HTML *interface* display application that runs through a J2EE container. Because of our requirement to support HTML thin clients, our *interface* widget set is somewhat limited compared to another widget kit, such as Java Swing. However, the event model (described below) and relationship of "high-level" to "low-level" widgets allow a significant degree of interface customizability even with the limitations of HTML. Other technologies, such as JavaScript and streaming video are presently being used to supplement our *interface* standard. Future *interface* display applications are under consideration, such as Unreal Tournament for Warrior Tutoring [9] (an entirely different domain, unrelated to our mathematics project), and Macromedia Flash™ for rich content definition.

The *behaviors* for each *problem* define the results of actions on the *interface*. An action might consist of pushing a button or selecting a radio button. Examples of *behavior* definitions are state graphs, cognitive model tracing, or constraint tutoring, defining the interaction that a specific *interface* definition possesses. To date, state graph or pseudo-tutor definitions have been implemented in a simple XML schema, allowing for a rapid development of pseudo tutors [13]. We have also implemented an interface to the JESS (Java Expert System Shell) production system, allowing for full cognitive model *behaviors.* A sample of the type of cognitive models we would wish to support is outlined in Jarvis et al [6]. The abstraction of *behaviors* allows for easy extension of both their functionality and by association their underlying XML definition.

Upon user interaction, a two-tiered event model is used to respond to that interaction. These tiers correspond to the two levels of widgets described above, and thus there are "high-level" actions and "low-level" actions. When the user creates an event in the *interface*, it is encoded as a "low-level" action and passed to the "high-level" *interface* widget. The "high-level" *interface* widget may (or may not) decide that the "low-level" action is valid, and encode it as a “high-level” action. An example of this is comparing an algebra text field (scripted with algebraic equality rules) with a normal text field by initiating two “low-level” actions such as entering “3+3” and “6” in each one. The algebra text field would consider these to be the same “high-level” action, whereas a generic text field would consider them to be different “high-level” actions. “High-level” actions are processed by the interpreted *behavior* and the *interface* is updated depending on the *behavior’s* response to that action. The advantage of “high-level” actions is that they allow an *interface* widget or content developer to think in actions relevant to the widget, and avoid dealing with a large number of trivial events.

1.1.3 Strategy Unit

The *strategy* unit allows for high-level control over *problems* and provides flow control between *problems.* The *strategy* unit consists of *tutor strategies* and the *agenda*. Different *tutor strategies* can make a single *problem* behave in different fashions. For instance, a scaffolding *tutor strategy* arranges a number of *problems* in a tree structure, or scaffold. When the student answers the root *problem* incorrectly, a sequence of other *problems* associated with that incorrect answer is queued for presentation to the student. These scaffolding *problems* can continue to branch as the roots of their own tree. It is important to note that each *problem* is itself a self-contained *behavior*, and may be an entire state graph / pseudo-tutor, or a full cognitive tutor.

Other types of *tutor strategies* already developed include message strategies, explain strategies, and forced scaffolding strategies. The message strategy displays a sequence of messages, such as hints or other feedback or instruction. The explain strategy displays an explanation of the problem, rather than the problem itself. This type of *tutoring strategy* would be used when it is already assumed that the student knew how to solve the problem. The forced scaffolding strategy forces the student into a particular scaffolding branch, displaying but skipping over the root *problem*.

The concept of a *tutor strategy* is implemented in an abstract fashion, to allow for easy extension of the implementation in the future. Such future *tutor strategies* could include dynamic behavior based on knowledge tracing of the student log data. This would allow for continually evolving content selection, without a predetermined sequence of *problems*.

This dynamic content selection is enabled by the *agenda*. The *agenda* is a collection of *problems* arranged in a tree, which have been completed or have been queued up for presentation. The contents of the *agenda* are operated upon by the various *tutor strategies*, selecting new *problems* from *sections* (possibly within *sections*) within a *curriculum* to append and choosing the next *problem* to travel to [4].

1.1.4 Logging Unit

The final conceptual unit of the XTA is the *logging* unit with full-featured relational database connectivity. The benefits of logging in the domain of ITS have been acknowledged, significantly easing data mining efforts, analysis, and reporting [11]. Additionally, judicious logging can record the data required to replay or rerun a user’s session.

The *logging* unit receives detailed information from all the other units relating to user actions and component interactions. These messages include notification of events such as starting a new curriculum, starting a new problem, a student answering a question, evaluation of the students’ answer, and many other user-level and framework-level events.

Capturing these events has given us an assortment of data to analyze for a variety of needs. User action data captured allows us to examine usage-patterns, including detection of system gaming (superficially going through tutoring-content without actually trying to learn) [4]. This data also enables us to quickly build reports for teachers on their students, as well as giving a complete trace of student work. This trace allows us to replay a user's session, which could be useful for quickly spotting fundamental misunderstandings on the part of the user, as well as debugging the content and the system itself (by attempting to duplicate errors).

The *logging* unit components are appropriately networked to leverage the benefits of distributing our framework over a network and across machines. The obvious advantage this provides is scalability.

1.1.5 System Architecture

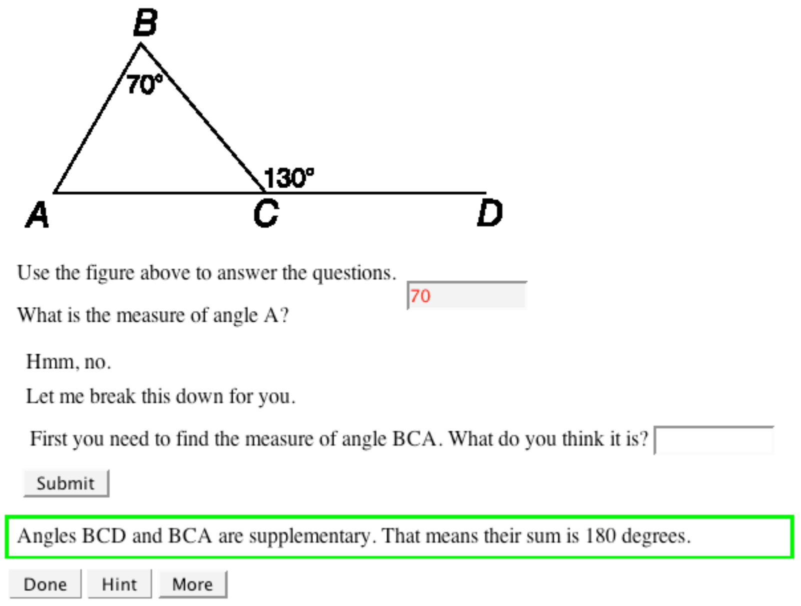
The XTA provides a number of levels of scalability. To allow for performance scalability, care was taken to ensure a low memory footprint. It is anticipated, based on simple unit testing, that thousands of copies of the XTA could run on a single machine. More importantly, the individual units described above are separated by network connections (see Figure 2). This allows individual portions of the XTA to be deployed on different computers. Thus, in a server context, additional capacity can be added without software modification, and scalability is assured.

The runtime can also transform with little modification into a client application or a server application instantiated over a web server or other network software launch, such as Java WebStart. Both types of applications allow for pluggable client interfaces due to a simple interface and event model, as described in the *interface* unit. A client side application contains all the network components described above (Figure 2) as well as content files required for tutoring, and has the capacity to contact a remote *logging* unit to record student actions. Running the XTA in a server situation results in a thin client for the user (at present either HTML or Java WebStart), which operates with the interface and event model of the server. Thus the server will run an instance of the XTA for every concurrent user, illustrating the need for a small memory footprint. The XTA instances on the server contact a centralized *logging* unit and thus allow for generated reports available through a similar server [4].

2. Methods and Results

The XTA has been deployed as the foundation of the Assistments Project [12]. This project provides mathematics tutors to Massachusetts students over the web and provides useful reports to teachers based on student performance and learning. The system has been in use for a year, and has had nearly 1000 total users. These users have resulted in over 1.3 million actions for analysis and student reports [4]. To date, we regularly support a live concurrency of approximately 50 users from Massachusetts schools. Additionally, during load testing, a single machine can serve over 500 simulated clients from a single J2EE / database server combination. The primary server used in this test was a Pentium™ 4 with 1 gigabyte of RAM running Gentoo Linux. Our objective is to support 100,000 students across the state of Massachusetts. 100,000 students divided across 5 school days would be 20,000 users a day. Massachusetts's schools have 7 class periods, which would be roughly equivalent to supporting 3,000 users concurrently. This calculation is clearly based on estimations, and it should be noted that we have not load tested to this degree.

Tutors that have been deployed include scaffolding state diagram pseudo-tutors with a variety of strategies (see Figure 3 for a pseudo-tutor in progress). We have also deployed a small number of JESS cognitive tutors for specialized applications. It should be noted that the tutors used in the scaling test described above were all pseudo-tutors, and it is estimated that a much smaller number of JESS tutors could be supported.



Use the figure above to answer the questions.

What is the measure of angle A?

Hmm, no.

Let me break this down for you.

First you need to find the measure of angle BCA. What do you think it is?

Angles BCD and BCA are supplementary. That means their sum is 180 degrees.

Figure 2 - Pseudo-tutor in Progress

In summary, the launch of the XTA has been successful. The configuration being used in the Assistments project is a central server as described above, where each student uses a thin HTML client and data is logged centrally. The software has been considered stable for several months, and has been enthusiastically reviewed by public school staff. Since September 2004, the software has been in use at least three days a week over the web by a number of schools across central Massachusetts. This deployment is encouraging, as it demonstrates the stability and initial scalability of the XTA, and provides significant room to grow.

3. Conclusions

The larger objective of this research was to build a framework that could support 100,000 students using ITS software across the state of Massachusetts. We are encouraged by our initial results from the Assistments Project, which indicate that the XTA has graduated from conceptual framework into a usable platform (available at http://www.assistment.org). However, this test of the software was primarily limited to pseudo-tutors, though model-tracing tutors are supported. One of the significant drawbacks of model-tracing tutors in a server context is the large amount of resources they consume. This resource consumption could be prohibitive in scaling to the degree that is described in our results without additional measures such as load balancing. Another partial solution to this might be the support of constraint-based tutors [10], which could conceivably take fewer resources, and we are presently exploring this concept. These constraint tutors could take the form of a simple JESS model (not requiring an expensive model trace), or another type of scripting language embedded in the state-graph pseudo-tutors.

Related research in our lab has yielded web-based systems for teachers to create and assign curriculums to their students, generate numerous reports on their students' strengths and weaknesses [4], as well as an application for rapidly authoring ITS content for use with the XTA [13]. Research and work in all of these areas is ongoing. Overall analysis of the system and its learning effects are covered in [12].

3.1 Future Work

Other planned improvements to the system include dynamic *curriculum* sections, which will select the next problem based on the student's performance (calculated from logged information). Similarly, new *tutor strategies* could alter their behavior based on knowledge tracing of the student log data. Also, new *interface* display applications are under consideration, using the *interface* module API. As mentioned, such interfaces could include Unreal Tournament™ (for applications such as Warrior Tutoring and other domains), Macromedia Flash™, or a Microsoft .NET™ application. All the components of the XTA were implemented in a modular and highly extensible way, so that adding new functionality is programmatically quite easy. We believe the customizable nature of the XTA could make it a valuable tool in the continued evolution of Intelligent Tutoring Systems.

References

[1] Anderson, J. R. (1993). Rules of the mind. Hillsdale, NJ: Erlbaum.

[2] Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, 4 (2), 167-207.

[3] Anderson, J.R., & Pelletier, R. (1991). A development system for model-tracing tutors. In *Proceedings of the International Conference of the Learning Sciences,* 1-8.

[4] Feng, Mingyu, Heffernan, N.T. (2005). Informing Teachers Live about Student Learning: Reporting in the Assistment System. *Submitted to the 12th Annual Conference on Artificial Intelligence in Education 2005, Amsterdam*

[5] Heffernan, N. T. & Croteau, E. (2004) Web-Based Evaluations Showing Differential Learning for Tutorial Strategies Employed by the Ms. Lindquist Tutor. *Proceedings of 7th Annual Intelligent Tutoring Systems Conference, Maceio, Brazil.* Pages 491-500.

[6] Jarvis, M., Nuzzo-Jones, G. & Heffernan. N. T. (2004) Applying Machine Learning Techniques to Rule Generation in Intelligent Tutoring Systems. *Proceedings of 7th Annual Intelligent Tutoring Systems Conference, Maceio, Brazil.* Pages 541-553

[7] Koedinger, K. R., Aleven, V., Heffernan. T., McLaren, B. & Hockenberry, M. (2004) Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration. *Proceedings of 7th Annual Intelligent Tutoring Systems Conference, Maceio, Brazil.* Pages 162-173

[8] Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, *8,* 30-43.

[9] Livak, T., Heffernan, N. T., Moyer, D. (2004) Using Cognitive Models for Computer Generated Forces and Human Tutoring. *13th Annual Conference on (BRIMS) Behavior Representation in Modeling and Simulation. Simulation Interoperability Standards Organization. Arlington, VA. Summer 2004*

[10] Mitrovic, A., & Ohlsson, S. (1999) Evaluation of a Constraint-Based Tutor for a Database Language. *Int. J. on Artificial Intelligence in Education* 10 (3-4), pp. 238-256.

[11] Mostow, J., Beck, J., Chalasani, R., Cuneo, A., & Jia, P. (2002c, October 14-16). Viewing and Analyzing Multimodal Human-computer Tutorial Dialogue: A Database Approach. *Proceedings of the Fourth IEEE International Conference on Multimodal Interfaces (ICMI 2002), Pittsburgh, PA*, 129-134.

[12] Razzaq, L., Feng, M., Nuzzo-Jones, G., Heffernan, N.T., Aniszczyk, C., Choksey, S., Livak, T., Mercado, E., Turner, T.E., Upalekar. R, Walonoski, J.A., Macasek. M.A., Rasmussen, K.P. (2005) The Assistment Project: Blending Assessment and Assisting. *Proceedings of the 12th Annual Conference on Artificial Intelligence in Education 2005, Amsterdam*

[13] Turner, T.E., Macasek, M.A., Nuzzo-Jones, G., Heffernan, N..T, Koedinger, K. (2005). The Assistment Builder: A Rapid Develoment Tool for ITS. *Poster, 12th Annual Conference on Artificial Intelligence in Education 2005, Amsterdam*