DEVELOPING A COGNITIVE RULE-BASED TUTOR FOR THE ASSISTMENT SYSTEM

By

Kai Rasmussen

A Thesis

Submitted to the Faculty

Of

WORCESTER POLYTECHNIC INSTITUTE

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

In

Computer Science

December 2006

Approved:

_____
Professor Neil Heffernan, Thesis Advisor

_____
Professor Dave Brown, Thesis Reader

_____
Professor Michael Gennert, Head of Department

Abstract

The ASSISTment system is a web-based tutor that is currently being used as an eighth and tenth-grade mathematics in both Massachusetts and Pennsylvania. This system represents its tutors as state-based "pseudo-tutors" which mimic a more complex cognitive tutor based on a set of production rules. It has been shown that building pseudo-tutors significantly decreases the time spent authoring content. This is an advantage for authoring systems such as the ASSITment builder, though it sacrifices greater expressive power and flexibility. A cognitive tutor models a student's behavior with general logical rules. Through model-tracing of a cognitive tutor's rule space, a system can find the reasons behind a student action and give better tutoring. In addition, these cognitive rules are general and can be used for many different tutors. It is the goal of this thesis to provide the architecture for using cognitive rule-based tutors in the ASSITment system. A final requirement is that running these computationally intensive model-tracing tutors do not slow down students using the pseudo-tutors, which represents the majority of ASSISTment usage. This can be achieved with remote computation, realized with SOAP web services. The system was further extended to allow the creation and implementation of user-level experiments within the system. These experiments allow the testing of pedagogical choices. We implemented a hint dissuasion experiment to test this experimental framework and provide those results.

Table of Contents

1. Introduction

Intelligent Tutoring Systems (ITS) can be defined as the use of computer tutors, which mimic one-on-one human tutoring. Through the use of artificial intelligence the ITS community hopes to achieve, through the use of computer systems, tutors that are as effective as human tutors. The term Intelligent Tutoring Systems covers a wide range of possible computer-based tutors, from cognitive model tracing tutors [6], constraint-based tutors [14], to pseudo-tutors. A pseudo-tutor is a simplified cognitive model based on a state graph. State graphs are finite graphs with each arc representing a student action, and each node representing a state of the problem interface[10]. Cognitive model tracing tutors use models of possible student actions to infer the reasoning behind a student's answer. Through this model, the system can build effective tutoring. Constraint based tutors use a series of constraints to comprise a problem. A student's input is checked against these constraints. An action is considered correct until it violates a constraint.

The ASSISTment system is a web-based ITS used within schools throughout central Massachusetts [20]. Its goal is to both *assist* students while *assessing* them. This systems primary mission is to provide practice for 8^{th} and 10^{th} grade student for the MCAS state-wide assessment exam. Currently researchers working on this project are exploring how to best extend the ASSISTment system to allow for new and potentially more effective tutor types. In order to do this, a system must be put in place to allow for testing pedagogical choices made in development. Testing of these pedagogical choices can be made through random experimentation of the behavior of the system's runtime behavior.

This thesis will explore the extensibility of the ASSISTment system. First the types of supported tutors will be discussed. This thesis will explain how the ASSISTment system was extended to allow cognitive tutors. Further extensions will be described through the use of web services. Finally, this thesis will describe an experiment framework for testing the ASSISTment system's pedagogical choices. Through this experimentation, the effectiveness of one runtime behavior over another can be discovered. This is a needed tool in the development of any ITS and is especially useful when designing

extensions to your system. An experiment which used this framework will be used to show its functionality and usefulness.

1.1 Thesis Objectives

The objective of this thesis is to extend an existing Intelligent Tutoring System to allow tutors that are more expressive. In order to achieve this I first looked at the different types of tutors available and compared them to the supported tutors of the ASSISTment system, a web-based math tutor. My first goal was to extend this system to allow the use of more expressive cognitive rule-based tutors. Currently the ASSISTment system and builder only support pseudo-tutors, which use no AI techniques. Pseudo-tutors are emulations of cognitive tutors. Cognitive tutors are tutors based upon a student model, realized through some production rule set. It was my goal to extend the current system to support the use of cognitive tutors as well as pseudo-tutors.

Secondly I set about putting in place the means for easily extending the ASSISTment system even further. This was accomplished with web services. I will show how web services can be used to increase an ITSs extendibility, as well as foster inter ITS collaboration.

Lastly, I put in place a framework for testing the design choices made within an ITS. There are a myriad of pedagogical choices to be made while designing an Intelligent Tutoring system. For example, a designer must make concrete choices on how and when to provide tutoring intervention to a student. Often research questions arise surrounding these design choices. We are proposing an architectural framework for dynamically changing the runtime behavior of these systems. With the proper controls, a researcher would be able to experimentally determine if their pedagogical choices are more effective then others as well as determine their tradeoffs.

In this experiment I aimed to determine if dissuading students from requesting help too quickly would affect the time student's spent reading hint messages. Students have many approaches to gaming Intelligent Tutoring Systems. One possible gaming method that

students have found in our system is requesting assistance quickly until they reach a 'bottom-out' hint where the answer is revealed. Students who request help unusually quickly are disengaged and are found to achieve less learning. By dissuading this behavior I found that students spent longer reading hints. I found that this effect is more significant when more obvious gamers are dissuaded. With more aggressive styles of dissuasion, we found a negative correlation and students spent less time on hints.

2. ASSISTment System

The ASSISTment system is a web-based algebra tutor that teaches 8th and 10th grade math in the state of Massachusetts. The system is used by students to prepare for the MCAS state assessment test (a test required to graduate). The goal of the ASSISTment project is to provide both *assistance* as well as *assessment* to students. It has been shown that, in addition to classroom instruction, the ASSISTment system has a significant affect on student learning [8]. In addition, we have built a range of reporting tools to provide accurate assessment for teachers.

Content is given to students in a manner similar to the MCAS exam. First students are given an original question and then, if a student provided an incorrect answer, they are given follow-up scaffolding questions, which provide directed line of reasoning on the original question's core skills. Assistance is given to students in both the form of these scaffolding questions and bug messages relating to common incorrect answers. Figure 1 shows an example of what the ASSISTment system looks like for a student. Above you see an original problem, followed by a scaffolding problem and some requested hint messages.

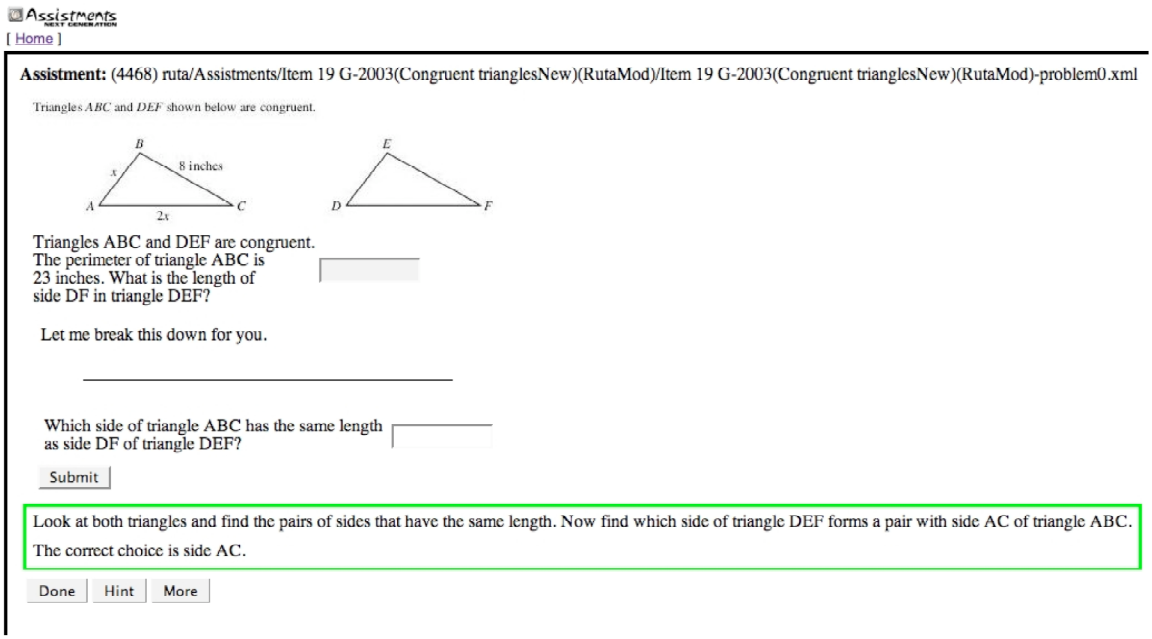


Figure 1: ASSISTment System

2.1 Extensions to ASSISTment System

This thesis is broken into three main sections, each describing an extension I made to the ASSISTment system. Each section has an individual background and results section as well as sections describing the work I've done in that area. First I will discuss extending the ASSISTment system to include cognitive Model Tracing tutors. Next, I will discuss implementing web services to extend the system even further, as well as deal with some performance issues of Model Tracing tutors. Lastly, I will discuss a framework I developed for this system for testing design choices. I will discuss a test I implemented using this framework and it's results.

3. ASSISTment Architecture

In this section, we will briefly discuss the ASSISTment runtime architecture, explaining its strengths and limitations. The ASSISTment system is a web-based math tutor built as a Java 2 Enterprise Edition (J2EE) application and was designed to assist students, using scaffolding questions and hint messages, as well as to assess students, through reporting pages and data analysis. Kodanganallur et al describes the high level ITS architecture as a system that contains a problem domain with tutor remediation through a student model component [12]. The student model could be a set of constraints in the case of a Constraint Based tutor, a set of production rules in the case of a Model Tracing tutor, or a state machine in a pseudo-tutor. The abstract view of this architecture, shown in Figure 2, has a wrapper around the Student Model to control tasks such as problem selection. The student's actions are inputted through the user interface and evaluated against the student model. The student model would then offer tutoring back through the runtime engine and to the user interface.

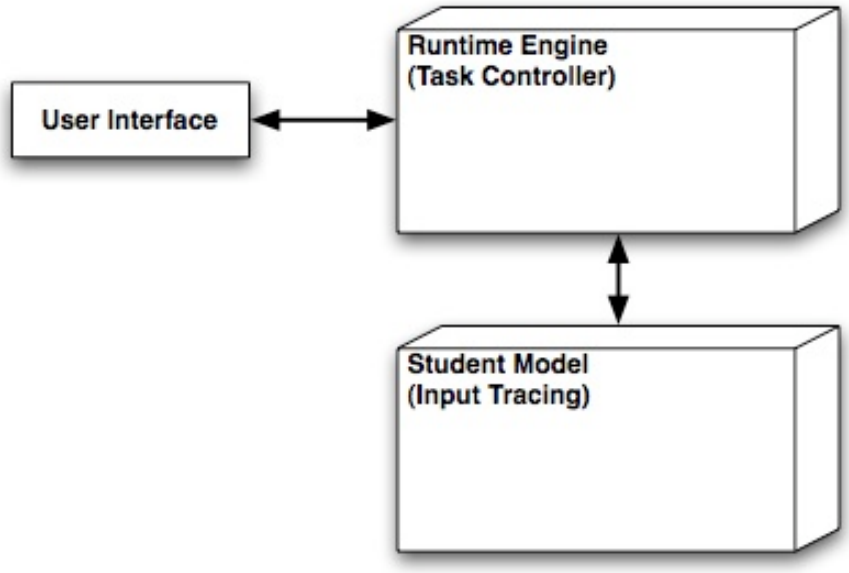


Figure 2: High Level ITS Architecture

The backbone of the ASSISTment system is the eXtendable Tutor Architecture [19]. This architecture can be conceptually broken into a runtime engine component and a student model component. In current literature this has been called the outer loop and

the inner loop of an ITS architecture [25]. The runtime engine (outer loop) is responsible for task selection while the student model (inner loop) is concerned with evaluation of student input from within a task and for providing feedback.

The runtime engine is composed of a *curriculum* component and a dynamic *agenda stack*. Content is rooted in curriculum components, which represent a series of *problems*. The curriculum is composed of one or more *sections*, with each *section* containing *problems* or other *section*. The *agenda* controls the ordering of problems outside of the curriculum and the order of tutoring. Problems contain *strategies* that can change the agenda. This provides an innovative dynamic staging of problems.

The student model is comprised of *problem* objects. The problem component represents a problem to be tutored, including questions and answers required to solve the problem. Each of these questions are 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. The *behaviors* for each problem define the results of actions on the interface

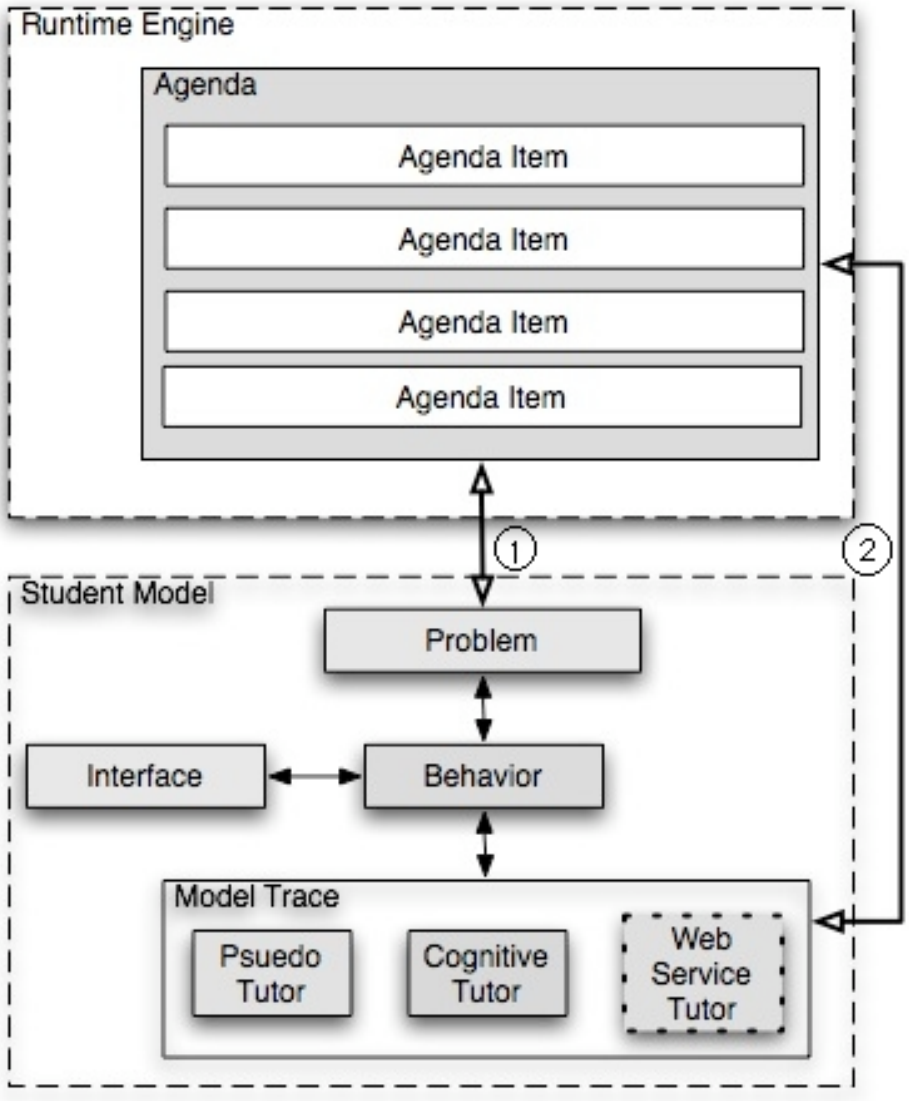


Figure 3: ASSISTment Architecture

The ASSISTment system roughly follows the same architecture as the inner loop/outer loop model. Figure 3 shows a more detailed view of this architecture. The backbone of the ASSISTment system is built upon the Common Tutor Object Platform (CTOP) objects. The task manager component is built upon an agenda stack. The agenda controls the ordering of problems and tutoring objects such as scaffolding, buggy messages, and hint messages. The runtime receives tutoring messages from the student model component, and uses to dynamically change the agenda.

Once a student's input is passed through the runtime engine, it is sent to the current student model on the top of the agenda. This student model is problem component, which represents a problem to be tutored, including questions and answers required to solve the problem. Each of these questions are represented by a problem composed of two main pieces: an interface and a behavior. The interface represents the problem's text, images, and various inputs while the behavior contains a model-tracing engine. The vast majority of problems within the ASSISTment system contain a pseudo-tutor student model.

Pseudo tutors are emulations of full cognitive tutors realized through a state machine. Each state represents a step in a problem and the transitions from one state to the next represent student input. Transitions can be marked as incorrect or correct and can be tagged with hint or buggy messages. These tutors can be created without AI and cognitive science training and are much less expensive to run.

4. Model Tracing Tutors

4.1 Background

In this section I will describe different tutor types including cognitive tutors such as Model Tracing and Constraint Based tutors as well as pseudo-tutors. I will then make a comparison of these tutor types and draw conclusions about their strengths and weaknesses.

4.1.1 Cognitive Tutors

When talking about Intelligent Tutoring systems, one is describing a system that uses artificial intelligence techniques to bring computer-aided instruction to the level of human tutors. The "intelligence" is the use of cognitive theory in order to develop a model of these human tutors. The theory is that through the use of these cognitive models, computer's can best simulate one-on-one human tutors. These models are typically based on a cognitive theory such as ACT-R [1] or SOAR [28], and realized by a production rule system.

A fundamental principle of the ACT-R theory is that "Acquiring cognitive knowledge involves the formulation of thousands of rules relating task goals and task states to actions and consequences." This statement introduces the idea that all knowledge can be represented as either syntactic or general world knowledge. All information that can be verbally described is called declarative knowledge while all information that can be inferred from behavior or actions is called procedural knowledge. For example, consider the knowledge that composes multicolumn addition. Some pieces of declarative knowledge would be "There is a column that has not been added" and "in that row there are two digits and no carry". The procedural knowledge would be "In that column, add those two digits. If the sum is greater than 10, then put the carry in the next column".

Another conclusion of the ACT-R theory states that all cognitive skills are realized productions rules. This simple statement implies that all human actions can be described

by a set of rules, though perhaps a uniquely complex one. An example of a production rule is this abstract algebra rule:

IF goal is to find an angle in an isosceles triangle ABC
and AC = AB and angle A is known
THEN set the value of angle B to A

It would be said that the knowledge of the existence of the isosceles triangle, and knowing the equality of the side lengths are all declarative knowledge. Knowing to set the value of an angle in this instance is procedural knowledge. All of this combined is a production rule.

One type of tutor that is based on cognitive models is a 'model-tracing' [6] [2] tutor, which follows a learn-by-doing approach. These tutors represent the knowledge domain as a series of production rules using some cognitive theory, with each production rule representing an implicit or direct action that can be performed. It is important to note that these productions are generalized based on the declarative knowledge state. The student 'learns by doing' by taking actions within the tutor's interface. The tutor will then track a student's progress through the series of production rules. This is called model-tracing. Upon receiving an action, the tutor can search the set of all possible actions and determine a series of productions that matches the student model. It is then possible to determine if it was a correct action or an incorrect action. On the case of an incorrect action the tutor has a detailed record of the reasons for that action (the set of followed productions) and can give feedback.

4.1.2 Constraint-based Tutors

Another cognitive approach is constraint-based tutors [14]. These tutors allow a student more freedom to their actions but impose subtle constraints to their inputs. An example of this is allowing a student to input any number as long as it is prime and smaller than 100. The next input would also follow another set of constraints. The student will work towards some problem goal, and will receive tutoring if they violate any constraints.

Constraints in these tutors are comprised of a pair of values. These values are the relevancy constraint (Cr) and the satisfactory constraint (Cs). Constraint satisfaction matches the formula: IF Cr THEN Cs. If in a problem step a Cr is true and a student enters a value, which invalidates Cs, then the student is said to be incorrect. Otherwise, a student is free to perform any action. An example of one of these constraint pairs is as follows:

> **C_r : A Base angle of an isosceles triangle is known as X and the student has calculated the size of the other base angle as Y**
> **C_s: The size of Y is X**

If a student enters a size of Y, which is not equal to X when X is known, then the student's answer is said to be false and the tutor will provide feedback.

4.1.3 Pseudo-tutors

A full cognitive tutor provides a great deal of flexibility and generality for intelligent tutoring systems, though they are costly to build. They require a great deal of both domain knowledge and knowledge in artificial intelligence programming and cognitive theory, often at a PhD level. It has been shown that it typically takes 200 hours to author one hour of content [16]. Because of this there was a push to develop a simpler form of tutor that can match the same behavior of a cognitive tutor. This is called a 'pseudo-tutor' [10] because it emulates an intelligent tutor without requiring AI code.

Where in cognitive tutors a problem's steps are generalized among production rules that operate on a set of working memory, pseudo-tutors describe a problem with a set of states. Each state represents a step in a problem, i.e. the working memory, and the transitions from one state to the next represent student input. Transitions can be marked as incorrect or correct and can be tagged with hint or buggy messages. Figure 4 shows an example of what a pseudo-tutor state-graph looks like conceptually. This is a pseudo-tutor for an algebra multi-column addition problem.

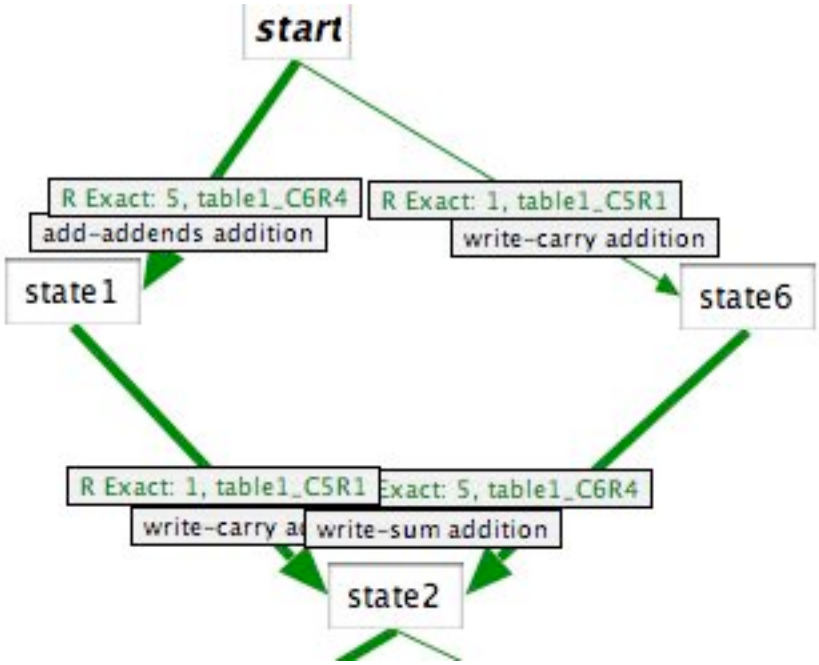


Figure 4: A Pseudo-tutor State Graph

The main advantage of pseudo-tutors is that they do not require any AI-knowledge to create. This allows domain experts to build tutors without expensive training. The Cognitive Tutor Authoring Tools application (CTAT) [10] builds pseudo-tutors by allowing the user to visually create the state diagram through the use of a behavior recorder. This recorder watches an author run through a problem, creating states at each action. The ASSISTment builder [24] creates pseudo-tutors implicitly and does not have any state editor. Because of this, the ASSISTment's pseudo-tutors are an even more generalized state graph.

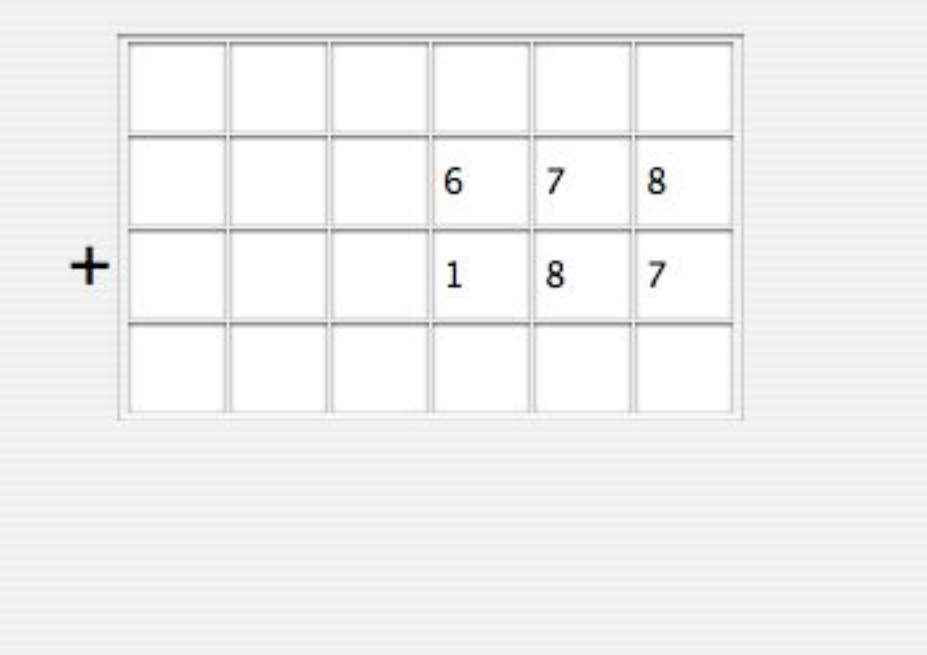


Figure 5: State Graph's interface

4.1.4 Comparison of Tutor Types

The strength of cognitive tutors is its flexibility, through production rules that generalize across problems and problem-solving states. These tutors allow subtle constraints on the ordering of problem steps as well as naturally allow many alternative paths through the problem state. This is also possible with a pseudo-tutor, but all alternative paths must be explicitly written. In a reasonably complex problem, the state diagram could become unwieldy. For example, the implementation of a constraint-based tutor would require states for each possible input of a constraint. The sub-states from these would only vary slightly, containing mostly duplication. These kinds of problems are native territory for cognitive tutors.

Cognitive tutors lend itself naturally to a more complex and flexible problem space. It is also important to note that because the ASSISTment builder only produces a simple state diagram, a full emulation of a cognitive tutor is not possible. The ASSISTment builder does not support problems with alternative inputs and paths through the problem state.

Model Tracing tutors are the most computationally expensive tutor of all the tutor types we have discussed. It requires a full search of a student model. Some would argue that this expensive a trade-off for it's strengths including generality and flexibility. Because it is based on a student model written by, hopefully, a domain expert, it is able to give very fine-grained tutor response on student input. Also, because it is able to solve a problem, the tutor is able to give planning advice to a student. One criticism is that Model Tracing tutors force a student to follow a fixed approach.

Constraint based tutors are much less efficient to run then Model Tracing tutors. They are limited by their constraint set. Because the model has no concept of a goal, it is not possible for the tutor to provide planning advice. The best equivalent constraint based tutors can offer is a list of all unsatisfied conditions and conditions that could be satisfied. A Model Tracing tutor will mark an answer incorrect if it cannot be found in the model, while a constraint based tutor treats all answers correct until proven incorrect. Because of this, if the constraint model is incomplete then it is possible for incorrect solutions to be treated as correct. This goes against the goal of Intelligent Tutoring Systems, which is to impose correct methods for solving problems and attacking misconceptions and possible faulty productions a student may have.

The big advantage of pseudo-tutors is that an author does not need any AI programming background in order to create content. Also, the time to author pseudo-tutors is significantly less then to create cognitive tutors. There has been some work in using automatic production rule generation [9] with the CTAT tools but they are not yet at useable state.

The addition of cognitive tutors to the ASSISTment system would provide ample benefit. It is true that pseudo-tutors can provide the same behavior as cognitive tutors, but the current ASSISTment builder cannot create pseudo-tutors of this complexity. Also, cognitive tutors are general and can be used among many different problems. These are our motivations for integrating full cognitive tutors into the ASSISTment system.

4.2 Implementation

Earlier, I have pointed out some weaknesses in the ASSISTment system. A trade-off was made early on in development to support pseudo-tutors to encourage rapid-development of student content. Here in year two of the project, there is a healthy database of eighth and tenth grade problems, with a regularly used builder to author content on a daily basis. The focus now is to extend the flexibility and expressiveness of ASSISTment tutors.

I extended the system to allow the basic authoring and assignment of full cognitive tutors. This will require extending the Behavior classes to allow a cognitive model to be used instead of the current state-based behavior. A production system must be chosen that both is computationally reasonable and operates on a strong cognitive theory. A model-tracing algorithm must be developed to search this model space while tutoring. A set of authoring tools must be built to author and edit these tutors within the ASSISTment system.

The Java Expert Shell System (JESS) is a forward-chaining production system founded on a cognitive theory much like ACT-R. Knowledge is stored as declarative "facts" and procedural "rules". The main difference between JESS and other cognitive theories is that is has no model for long-term and short-term memory, which is not a concern for the types of problems in our system. Our cognitive tutors will be written in the JESS production system.

Each CTOP unit comprises of interfaced components, and thereby naturally lends itself to extension. The Behavior interface provides the methods for performing hints and performing actions. I created a new JessRuleBehavior component, which implements the behavior interface. It contains a JESS engine for storing facts and rules. The perform action methods will run a model trace on the current state of the problem, using the incoming action from the student. Therefore a model-tracing engine must also be used by this new JessRuleBehavior to determine the correctness of a students input and provide strategies when an input is incorrect.

The model-tracing algorithm must create a full search tree of all possible actions from the current problem state. This includes possible buggy (incorrect) productions, which a student might have performed. Because this search space could become quite large, we limit our search to a certain number of productions. Also, some productions occur between explicit actions, such as "need to carry" occurs before "add with carry", so we will end a trace when a production that produces an action is reached. The average number of productions fired until an action is performed changes between tutors. Therefore, we perform an iterative deepening search on the number of productions to fire.

To test this new addition, I created a full cognitive tutor that worked within the ASSISTment system. This tutor was a multi-column addition tutor. It forces students through each step from choosing the right column to add, to making and adding carries from column to column. I choose this tutor because it is a classical example of a cognitive tutor, supported and implemented and many Carnegie Mellon University ITSs. I had to convert the typical interface into one supported by the ASSISTment system, write the JESS rules to perform multi-column addition within this interface, write typical buggy rules. I then had to import these rules into a JessRuleBehavior object that I created for the ASSISTment system. Figure 6 shows this addition tutor. The Appendix contains the JESS rules for this problem.

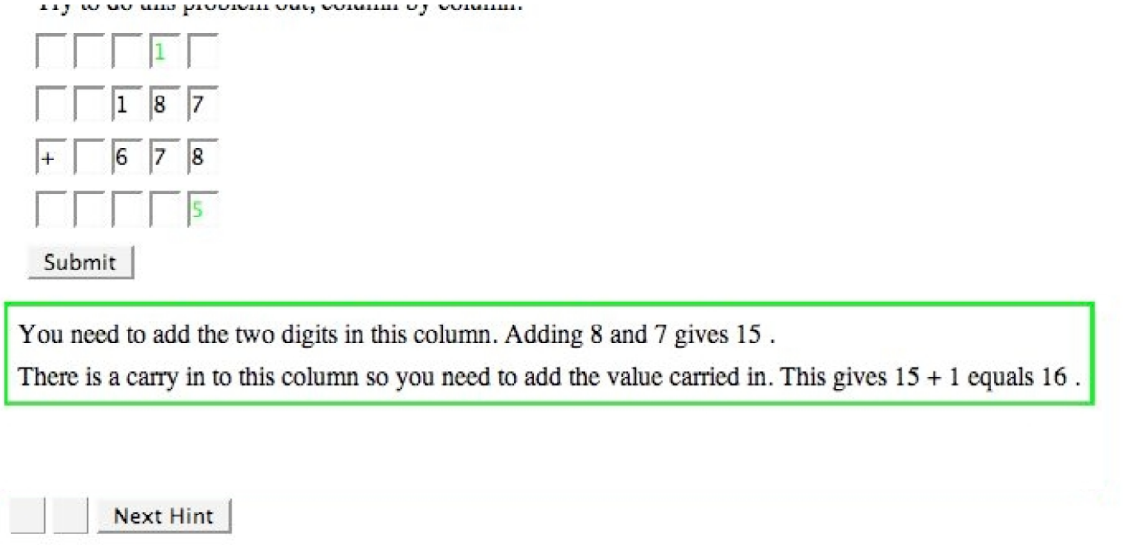


Figure 6: Addition Tutor

4.3 Results

The ASSISTment system now supports a full Model-Trace tutor based on a cognitive model. In the ASSISTment system, the Java Expert Shell System (JESS) rule engine powers our cognitive student model. The Model Trace algorithm is not a scalable solution for web-based tutoring systems. Through the use pseudo-tutors, the ASSISTment system has been estimated to support 600 concurrent users, though the system can only support about 20 concurrent users using the Model Tracing tutors. The result of this stress test is that if a single classroom of students are using a Model Tracing tutor, no other students can use the system, whether they use Model Tracing tutors or the normal pseudo-tutors.
In the next section we will look at one way of creating a scaleable solution for using cognitive tutors within a web-based ITS. The solution we found was by offloading the Model Tracing algorithm through web services. This would effectively isolate students running through cognitive tutors, leaving the performance uneffected for the majority of students using pseudo-tutors.

Also, I found because there were no tools to assist me in creating this tutor, authoring was a very difficult process. There are no tools outside of the ASSISTment builder for creating interface objects that can be used for problems. Also, testing and editing of this problem once the rules had been created was very difficult. As stated earlier, the ratio of time spent created a cognitive tutor versus tutoring time it offers is about 200:1, while the same ration for pseudo-tutors is about 20:1. I did not perform any assessment on the ratio to build cognitive tutors within the ASSISTment system (because it was all done by scratch by me with not tooling) but my estimate is a ratio of at least 200:1 or greater. The one benefit of cognitive tutors is that they are general over a large problem space. Once a tutor is created, it can be used many times. These problems will be unique, though isometric, and all use the same production rules.

5. Web Services

Existing Intelligent Tutoring Systems (ITS) are the cumulative effort of a community dedicated to the research question: can intelligent computer tutors successfully mimic human tutors? There have been many approaches such as cognitive Model Tracing tutors [2][6] and Constraint-Based tutors [14] implemented in various languages, with highly varying pedagogical methods and interfaces. With a strong community it would seem beneficial for these researchers to be able to work together by using content and tutoring methods from within other existing systems. With this collaboration, tutors can reach broader audiences. The question remains: how do two seemingly different tutors cohabitate peacefully? This paper will define a method for achieving this through remote computation implemented with web services. This section will also show the benefits of web services, such increasing a system's scalability and extendibility.

We will briefly described remote computation and some of its benefits in terms of Intelligent Tutoring Systems and provide some examples of how web services can be used to extend systems. In the next sections we will discuss SOAP web services and their implementations in more detail. Next, we will briefly describe a high level view of the ASSISTment architecture and then provide a detailed description of how it was extended to use web services in a first step towards a full Service-Oriented Architecture. Finally, we will discuss our results, in terms of successful extensions to the ASSISTment system using web services, and our conclusions.

5.1 Background

Remote computation is the ability to offload a piece of a tutor's runtime component to a separate system through distributed computation mechanisms. This computation could range from evaluation of student input to problem selection or even a request for content itself. This allows collaboration between two ITS researches and between an ITS researcher and content providers who do not have the background to create their own tutoring framework.

For example, there has been interest in using the front-end of the ASSISTment tutor to host content from fellow researches with unsupported tutor-types within the ASSIstment system. An example of such tutor-types is a programming language tutor, which is capable of compiling and evaluating Java or Scheme code. To achieve this new goal, structure would need to be implemented in the runtime to ensure that this service will not interfere with students working on ASSISTment content. This service must run at the collaborator's system to maintain performance and to ensure reliability. If the service, running on a collaborator's system, cannot be reached, or dies while in the middle of computation, the ASSISTment system (being used as a portal) could then give an appropriate error message stating that, for example, 'Prof. Heinemann's Java service is experiencing difficulties'. It would then allow the student to skip the problem. The ASSISTment project has also been approached by content providers with no ITS background. They wish to also use the ASSISTment's portal for their own content, which possibly cannot be built using our authoring tools. An implementation of remote computation would allow this as well.

Additionally, remote evaluation can be used to offload potentially expensive services to external systems to provide new features without the performance trade-off. Model Tracing is an example of one of these operations. Because Model Tracing tutors are more expensive to execute, they are a prime example of the usefulness of remote computation in an ITS. This is especially true in the ASSISTment system, where cognitive Model Tracing tutors are used significantly less often then faster pseudo-tutors. External evaluation of the Model Tracing algorithm will not affect the quality of service for the majority of students using pseudo-tutors.

These forms of remote evaluation can be implemented through web services. A web service is a platform independent application that is called over Internet protocols such as HTTP and is driven by XML data. Because of its platform independence, it is possible to create web service front-ends to new and legacy systems. A large application built using many web services is said to have a Service-Oriented Architecture.

5.1.2 SOAP Web Services

Web services are defined as a piece of logic hosted somewhere on the Internet, accessible by standard protocols such as HTTP, and based upon an XML standard for communication, data transfer, service discovery, and interface publishing [17][5]. Because XML is used for the data representation layer, it remains platform and technology independent.

Web services are powered by three main standards-based technologies: Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). SOAP is an XML packaging structure for web service communication, which defines an interoperable form of data transfer between web services. WSDL is a standard description of web service interfaces, including input and output representation of methods. It also describes the binding protocols needed for connecting to a service. This allows clients to understand how to communicate to discovered services. UDDI is a set of technologies for publishing and discovering published services. Figure 7 shows an example of how these technologies are used in tandem. A typical system would interrogate a UDDI registry for information about a given web service and obtain a WSDL of the discovered service. The client would then use this WSDL to communicate to the service using SOAP messages.

A system built upon using web services is said to employ a Service Oriented Architecture (SOA). The strength of SOA is having a very loosely coupled system built upon independent and replaceable components. The web services that compose SOA can be executed without knowledge of the language that the service is written in or even where the service will run.

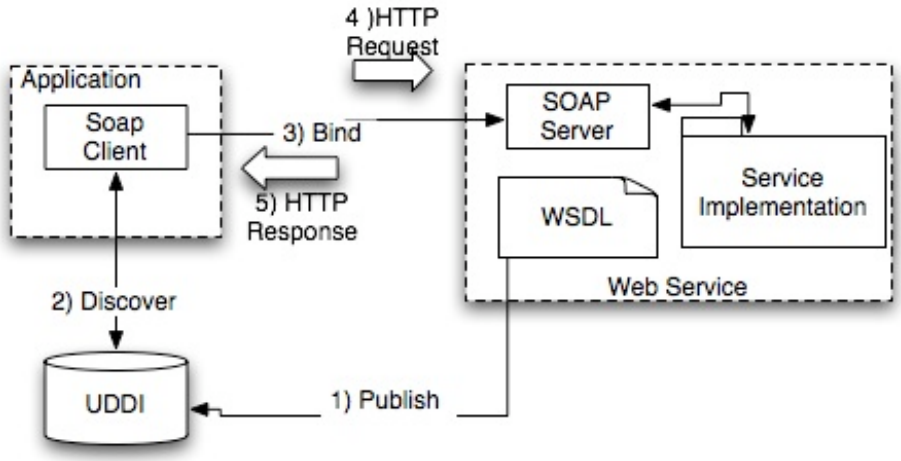


Figure 7: Typical Web Service Architecture

5.2 Implementation

In this section we will describe the process of implementing web services within the ASSISTment system. Because our goal was to extend the student model for remote evaluation of input, the behavior module was the choice for extension. In this new architecture, the student input is sent to a SOAP web service and a tutor response is sent as a reply. A generic, architecture general interface is essential to ensure cross-tutor collaboration.

We added two new behavior types; a state-less web service behavior and a state-full web service behavior. The main difference is the state-full services maintains a full student model and can therefore provide tutoring over an entire problem where the state-less service can only provide tutoring over a single problem state.

The state-less service has no record of a tutoring 'session' and represents a single step in a full problem. It's implementation is based upon the ASSISTment pseudo-tutor's state machine but it's interface is generic enough to provide a wide range of tutoring. We will show how it was used for pseudo tutors as well as a tutor on the programming language Scheme. We hope to expand these services in the near future. Through XML communication, the web service accepts a unique identifier of the current problem step

and the student action. This action describes a student input and action type. The web service's response is a generic strategy, which represents the tutor response to this input. This strategy could be a bug message, a scaffold, or a uniquely new strategy type. The web service also allows the problem step to be interrogated for available hints.

State-full web services keeps a problem 'session' and can therefore represent an entire problem or set of problems. Runtime engine's that wish to use this state-full web service as their student model would be required to start a session and use that session in all subsequent communications to the web service. This is the main difference between these two web service types as both state-full and state-less services accept student action and return tutor strategies. Use of a problem session allows more complex tutors to be used in a web service. For example, the JESS Model Tracing tutors cannot be fully described without this state-full service.

Because the interfaces for these web services are platform and implementation independent, they support a wide variety of uses. The client code for these services where easily incorporated into the ASSISTment system by extending our student model component. Since SOAP web services are built upon a standard, external tools exist that consume WSDL descriptions of web services and generate skeleton code for both the service and the client. Example of these tools are the Eclipse Web Tools Platform [7] and the Apache AXIS project [3]. Once the client code is in place in an ITS, a myriad of new tutor types are available and the systems can achieve new levels of extendibility.

5.3 Web Service Extentions

In this section we will discuss three new extensions to the ASSISTment system using a web service based student model. As a proof of concept we developed a services for ASSISTment pseudo-tutors, which mimics our runtime architecture. Next the JESS Model Trace tutor was implemented as a web service to help scale the model tracing algorithm. Finally, we've developed an entirely new tutor type that the ASSISTment

system cannot currently support in order to show how web services can improve extendibility and collaboration between intelligent tutoring systems.
The first tutor that implemented our web service student model was a replication of the ASSISTment pseudo-tutor. This service used the state-less web service and is offered as a way to host ASSISTment content to external ITSs as well as serve as a proof of concept of web services usefulness.

As previously stated, our largest motivation for moving to SOA was the ability to run Model Tracing tutors without effecting the quality of service for the majority of students being tutored with our pseudo-tutors. With two full classrooms running a JESS tutor, our system starts to hit a bottleneck with our current server setup. With the Model Tracing engine running as a web service we have increased the number of simultaneous running students dramatically. This number can be increased even further if there are multiple services running and our runtime can load-share among them.

To show how web services can be used to provide additional extendibility and increased collaboration, we are developing an entirely new tutor previously unsupported by the ASSISTment system. At the time of writing this tutor is still in development but will soon be completed. We are working with the TeachScheme [23] program to provide a programming language tutor. The web service is wrapped around a scheme evaluator and is focused on teaching both syntax and proper coding procedures, such as test cases and pre-condition and post-condition contracts.

In the future we can foresee easily adding new types of collaborations. This includes running a Java tutor or hosting homework assignments from the physics department. Because these tutors are compartmentalized as web services, their runtime does not affect system performance of the ASSISTment system. We can also foresee wrapping a web service over the entire ASSISTment system, allowing all of our 8^{th} and 10^{th} grade math content available for us for other ITS.

6. Experimental Framework

Gaming behavior is the intentional misuse of an Intelligent Tutoring System in order for a student to proceed with as little effort as possible. Examples of this behavior include exhausting provided help and using unnecessary guesses. A designer of an ITS must walk a fine line between providing necessary tutoring for engaged students who actively need assistance when they reach an impasse, and not allowing easy exploitation of this help structure for gamers.

We wished to explore the effects of dissuading gaming behavior within our system. Recent experimentation has been performed, demonstrating that a simple delay on hint requests could dissuade students from requesting unnecessary help, and thus have a positive effect on learning [13]. We wondered if we could dissuade students in a similar way.

We implemented a dissuasion experiment that, through calculated reading rates of hint messages, prompted students to slow down and reread the previous hint. We tested the hypothesis that a dissuasion of this nature would slow students down while they are reading hint messages. In addition, we describe a framework that was built in our system for quickly implementing and controlling these kinds of randomized experiments.

6.1 Background

The use of experiments within an Intelligent Tutoring System can be used to discover the effects of changing the runtime behavior of that system. All systems must go through an iterative process of testing and evaluation. Through a process of evaluation, an ITS will evolve into more effective versions. Koedinger describes an ITS design as one that is motivated by specific pedagogical hypotheses and informed by user testing [11]. When user testing fails, it must lead first to redesign and possible change to the underlining theory. This 'experimental manipulation of tutor components' is described by Albert Corbett as a parametric evaluation of an Intelligent Tutoring System [6]. The types of

behavior these parametric evaluations target are mostly pedagogical in nature (such as feedback and content control).

6.1.1 Runtime Experimentation

Previously, in literature, effects of pedagogical design decisions were tested through the comparisons of separate systems. In an example of this, Jeff Rickel designed four independent pedagogical agents to discover the strengths of different approaches to task-oriented tutoring [22]. Douglas Merrill compared three distinct Lisp tutors to determine which types of scaffolding guidance leads to superior performance [15]. In addition, Koedinger compared distinct versions of a prototype algebra tutor to determine the effect of algebraic symbolization [11].

We are proposing an ITS architecture for performing such experiments without the need of comparing different systems. This architecture will provide the framework for dynamically changing the behavior of the systems runtime. These changes can be randomly controlled at a user level (behavior changes depending upon the running user) or at a problem level (behavior changes depending upon the problem being tutored).

6.1.2 Learning Effects of Scaffolding

Through the lifetime of the ASSISTment system, we've had positive results in the effectiveness of our tutor [20]. The question remained if students were performing well because the intelligent tutoring system's ability to give fine-grained tutoring, or was it just an effect of having more practice with math problems. Razzaq [21] ran an experiment over the ASSISTment system to help understand if our tutor was more effective then problem practice alone. Razzaq found through this experiment that students who received tutoring in form of scaffolding questions performed better then those who did not

The experiment Razzaq performed in order to determine the effectiveness of the ASSISTment scaffolding is an example of a pedagogical experiment. In this experiment, 11 MCAS items on probability were given to 8^{th} grade students. Some students were given

items, which contained hint and bug messages but no scaffolding where the other students received the scaffolding version. It was found that students that received the scaffolding questions did better on a post-test of related problems. This is considered to be a curriculum level experiment because the behavior change of the system was determined by which problem a student was assigned.

6.1.3 Gaming Behavior Prevention

Gaming within an ITS can be defined as exploiting the feedback and help mechanisms in order to complete a task with little or no work. It has been shown that gaming behavior leads to substantially less learning [4]. Within the ASSISTment system, gaming detection and prevention were recently explored using this experiment framework. Walonoski created profiles of student gamers through a combination of classroom observation, previous literature, and logged student data; these profiles included information about the speed of hints requested, number of hints given, and speed of guesses [26].

Figure 8: Passive Intervention

From this information, we conducted an experiment where the system intervened when it detected gamers; classes received active intervention, passive intervention, or no intervention at all. Passive intervention was provided through a chart, which was visible to students with a representation of their gaming score and effort within the system. This is shown if figure 8. Active intervention was providing through pop-up messages informing the student that they were gaming. This is shown in figure 9. It was concluded that active intervention was effective at reducing gaming behavior [27]. This is considered a student level experiment because the behavior of the system changed depending on the student that was using the system—not because of the problem currently being worked on.

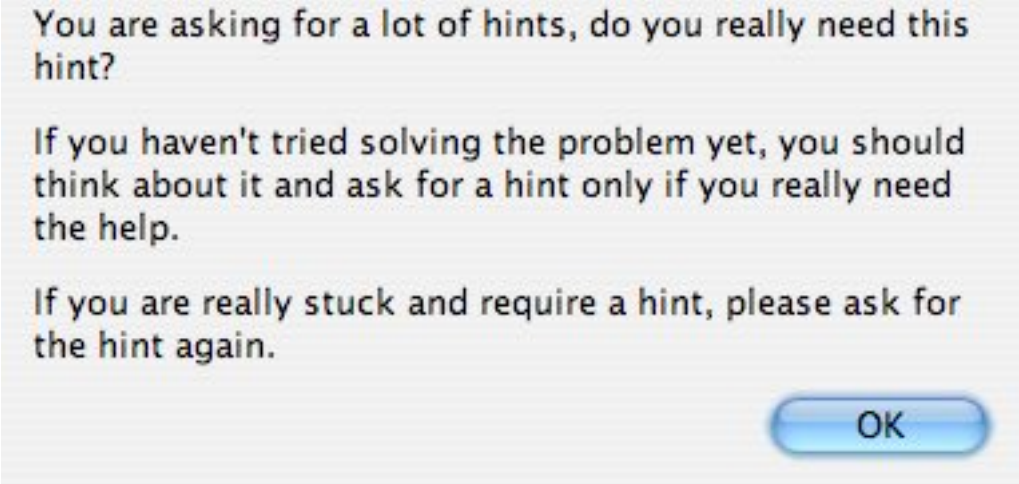


Figure 9: Active Intervention

6.2 Experimental Framework

In this section, we will discuss the proposed framework within the ASSISTment system that allows for that addition of controlled experiments and provides the means for parametric evaluation of our system. We will provide examination of both experiments run at the curriculum level and experiments run at the user level.

6.2.1 Curriculum Level Experiments

Curriculum level experiments are run with a special type of curriculum section. This section contains many different sets of problems that a student could receive. On entering an experimental section for the first time, the student is randomly assigned one set of problems from the possible sets. This experimental curriculum will typically contain a set of problems, which are designed to answer a research question, and a normal control set of problems. This experiment section is also typically book-ended with a shared pre- and post-test section for correlation.

Once an experiment has been completed, an analysis tool is used for mining data related to the curriculum and the experiment. This tool collects data for all students who completed the experiment curriculum. Typical data includes performance on each completed problem

and how much time spent on each problem. The experiment analysis tool is also able to automatically compare a student's performance on particular items and sections

6.2.2 User Level Experiments

The differentiations between user level experiments and curriculum level experiments include a difference in the code being run for each student in the experiment; user level experiments are also assigned by a user basis and are not dependant upon the problem being tutored. Curriculum level experiments are conducted by changing a core element of the content that is presented to a student, and that content is encapsulated within an experimental curriculum section. In user level experiments, the underlying code branches for each experiment condition. This provides a dynamic change of the system's behavior at runtime. The gaming detection and prevention experiment previously mentioned is an example of a user level experiment.

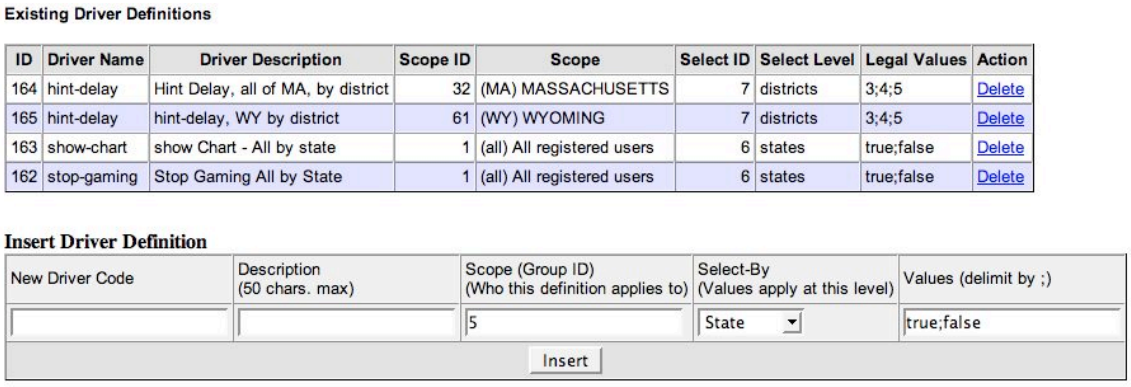
6.2.2.1 *User Profile*

Users within the ASSISTment system are organized within a hierarchy of groups. Students belong to a class group or a series of classes. Classes belong to a school group. School groups belong to a District, which belongs to a State, which belongs to everything. Because of this strict hierarchy, we can build a model of every group of each type to which a student belongs. User level experiments can be assigned to individual groups. The User profile is a collection of every experiment assigned to groups the student directly or indirectly belongs. This profile is used at runtime to determine which experiments to conduct for the student, as what values to assign to any experiment parameters.

6.2.2.2 *Experiment Driver and Assignments*

User level experiments are powered by an experiment driver. This driver specifies a driver-code, a scope, a select type, and a set of legal values for the experiment. The driver-code is a unique identifier for each experiment type, and is searched for in the user

profile during runtime at points where an experiment could be run. The scope is the group at which the experiment applies. For example, if an experiment has a scope of MA, the experiment would be activated in all of Massachusetts. The select type determines how the experiment will be controlled. If the select level is by class, then each class within the scope of the driver could have a different experiment value. This value controls the experiment; for instance, it could be as simple as true or false (in which case the experiment is either on or off), or it could be a list of permitted numbers. The tool, which can be used to control these experiment drivers, can be seen in figure 10.



Existing Driver Definitions

ID	Driver Name	Driver Description	Scope ID	Scope	Select ID	Select Level	Legal Values	Action
164	hint-delay	Hint Delay, all of MA, by district	32	(MA) MASSACHUSETTS	7	districts	3;4;5	Delete
165	hint-delay	hint-delay, WY by district	61	(WY) WYOMING	7	districts	3;4;5	Delete
163	show-chart	show Chart - All by state	1	(all) All registered users	6	states	true;false	Delete
162	stop-gaming	Stop Gaming All by State	1	(all) All registered users	6	states	true;false	Delete

Insert Driver Definition

New Driver Code	Description (50 chars. max)	Scope (Group ID) (Who this definition applies to)	Select-By (Values apply at this level)	Values (delimit by ;)
		5	State	true;false

Insert

Figure 10: The Experiment Driver Manager

An assignment is a particular instance of an experiment. Each assignment has an experiment value, from the driver's list of legal values, associated with it. To which groups get an assignment is determined by the select type. When an experiment driver is added, an assignment is created for each group of the select type under the scope of the experiment. The ASSISTment system will randomly give each experiment assignment a value from the driver's list of legal values. In this case, the creation of a driver effectively creates a random experiment. If randomness is not desired, the user can change these experiment values manually.

6.3 Hint Dissuasion Experiment

Gaming prevention has been a strong focus of the ASSISTment system. Some early work was done with the intervention experiment previously mentioned. One form of gaming possible within the ASSISTment system is the exploitation of our reactive help system. A

student can request hints on the current problem step at any time. In order to guarantee that student will not be 'stuck' on a problem, every series of hints includes a 'bottom-out' hint, which provides the current solution.

An example of gaming behavior with the hint system is quickly exhausting all available hint messages until the solution is provided. Gamers of this type spend little to no time engaged in the problem, and will therefore show little to no learning. We implemented an experiment to test if we could have these types of gamers spend longer on each hint, with the intent that the longer spent reading a hint message the more engaged the student will be with the help system.

This experiment is a slightly modified replication of a recently published experiment run by Charles Murray and Kurt VanLehn. In this experiment, they wished to correlate performance with the number of student hint requests [13]. Two groups of student were each given the same calculus tutor. In one class, the tutor provided a significant delay when a student requested a hint, and in the other group, there was no hint delay. The hypothesis of the experiment was that dissuaded students would ask for less *unnecessary* help and would therefore learn more. Their results stated that dissuaded students requested help less often and the number of help requests were negatively correlated with post-test scores.

Our experiment is slightly different from Murray and VanLen's. We wished to determine if dissuading 'speed-hinting' would cause students to spend more time reading hints. We did this by calculating a 'words-per-minute' reading rate on each hint request, using the number of words in a hint message, and the speed at which it took a student to ask for another hint. Problem difficulty and hint complexity were not taken into account. This reading rate is calculated for all students in the system for comparison against students who are run in this experiment. Instead of a hint delay, we decided to a use a different dissuasion tactic. If a student asks for a new hint faster then a set reading rate, the will system respond with the message, "You asked for a hint quickly. Have you read the last

hint?" rather then provide a new hint. This message will be repeated until a student slows their hint request rate.

For this experiment, we choose to aggressively penalize students for reading hints too quickly. A reading rate of 3 words-per-second was chosen as a maximum speed for reading hint messages. If a new hint was requested faster then this reading rate, the dissuasion message would be given to the student. One hundred and forty-eight students in various classrooms in the city of Worcester, MA conducted this experiment. Students were randomly selected into two groups. Either students were penalized with the dissuasion message or no form of dissuasion was used. Reading rates were calculated and logged for all students for comparison. Of the total number of students in the experiment, 75 received dissuasion messages and 73 did not.

6.4 Results

The following data represents an aggregation of our experiment's data. For each student we calculated his or her reading rate at a point where the student would have received our dissuasion. After completing the problem, we then calculated their reading rate when they next requested help. We have come to the following conclusions

Aggressive Dissuasion Discourages Active Students

Table 1: 3 Words-per-second Results

	Greater then 3 wps (Second attempt)		
Dissuaded	FALSE	TRUE	Grand Total
TRUE	28	47	75
FALSE	30	43	73
Grand Total	58	90	148

Of the 75 students who received a dissuasion message, 63% of these students continued to read hints faster then the desired rate. Of the 73 students who were not dissuaded, 51 % of the students continued to game the system. These results were weakly significant, with a p value of .64. This is the opposite effective then we wished would occur. This is shown in table 1.

We then attempted to understand this effect. We understood that three words-per-second would perhaps be too tight a definition of 'too fast'. If we accept 3 words-per-second as an aggressive rate, then we must accept that non-gamers are being penalized. To understand what the effect of our dissuasion on true gamers, we then looked only at students who demonstrated unusually fast hint reading rates.

Students who Read Hint Messages Faster are More Dissuaded

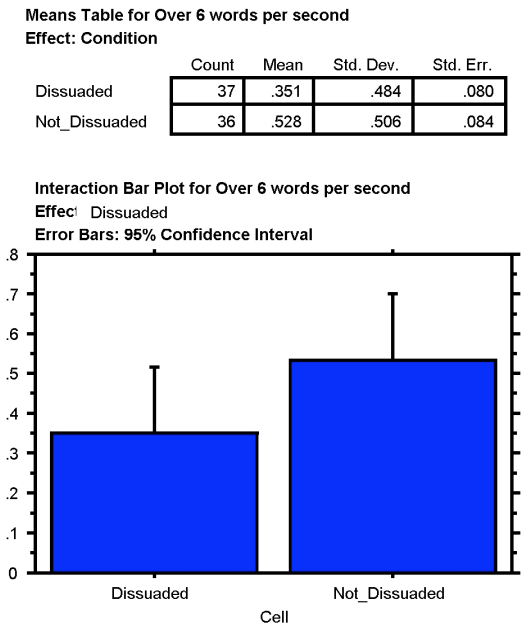


Figure 11: Mean Values for 6 Words per Second test

Table 2: 6 Words per Second Test Results

	Greater then 6 wps (Second Attempt)		
Dissuaded	FALSE	TRUE	Grand Total
FALSE	17	19	36
TRUE	24	13	37
Grand Total	41	32	73

We looked at our data again, reclassifying gamers as students who had a hint reading rate of greater then 6 words-per-second. This is a looser classification of hint gamers and became a more accurate assessment of the data. Of the 148 students who were classified as gamers under the aggressive 3 words-per-second rate, 73 were classifiable under the new 6 words-per-second rate. The data shows that 64% of student's who were given

dissuasion messages and had a reading rate greater then 6 words-per-second, slowed down below this classification. Only 47% of the students who did not receive the dissuasion slowed their reading rate to below this new classification. This result is mildly significant with a p-value of .134. This is shown in Figure 11 and table 2.

Following this trend we looked at students would were classified as gamers with a reading rate greater then 8 words-per-second. We found an even greater dissuasion effect for these students with unusually fast hint reading rates: 73% of the dissuaded students slowed down below this classification while only 56% of the non-dissuaded students decreased their reading rates. Though this is a stronger support to our claim, the result is less significant with a p value of .2. This is shown in table 3 and Figure 12.

Table 3: 8 Words per Second Test Results

TooFast8First	TRUE

Count of STUDENT	TooFast8Second		
DUSSUADED2	FALSE	TRUE	Grand Total
FALSE	17	13	30
TRUE	19	7	26
Grand Total	36	20	56

Means Table for Over 8 words per second
Effect: Conditon

	Count	Mean	Std. Dev.	Std. Err.
Dissuaded	26	.269	.452	.089
Not_Dissuaded	30	.433	.504	.092

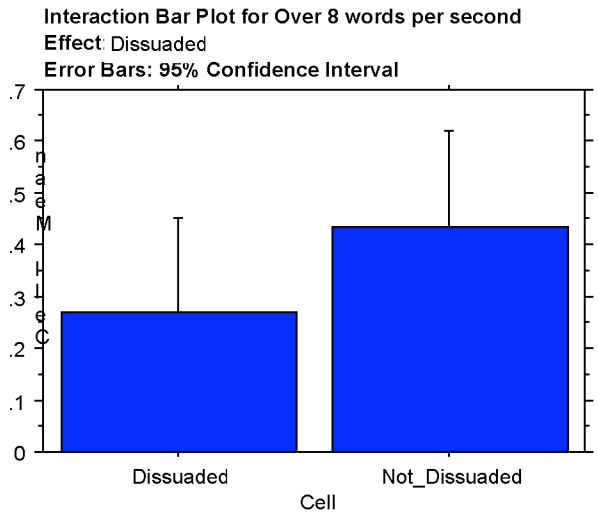


Figure 12: Use of 8 words per second as Classifier

7. Conclusions

The ASSISTment system allows the rapid development and deployment of pseudo-tutors. Cognitive tutors provide a more flexible, generalized tutor based on a production rule system. Cognitive tutors provide a more expressive tutor with the trade-off of increased development time and higher computational expense. One purpose of this thesis was to integrate cognitive tutors into the ASSISTment system. We were successful in extending the ASSISTment system to support full cognitive tutors implemented in the JESS production system. We performed stress testing of these tutors and found them much more expensive in terms of performance then pseudo-tutors. The ratio of simultaneous supported users for pseudo-tutors against cognitive tutors was about 10:1. We then investigated using web services in an attempt to make the use of cognitive tutors more scalable. Web services also allowed the ASSISTment system a higher level of extendibility and collaboration.

We have described the motivation and execution of changing the ASSISTment system to a Service Oriented Architecture. We showed that through its use we would be able to dampen the effect of running expensive Model Tracing in a web-based system. Also we demonstrated that SOA allowed the ASSISTment system to increase its extendibility. We illustrated these extensions with three new web service tutors. These services are platform and implementation independent which allows for a wide-variety of uses. Because they are open web services they offer a new form of collaboration in the ITS field. Through the offloading of expensive forms of evaluation, such as model tracing, were able to increase our systems scalability. We hope to expand our use of web services even further in the near future with new tutor types and collaborations.

Design of an Intelligent Tutoring System is an iterative process. Pedagogical choices in the design should be constantly evaluated and tested through experiment. If the test failed to match the theory that represents the design choice that was made, a system should go through redesign. In this paper, we have proposed an architectural framework for designing

and implementing these types of experimentation. This framework was then tested with an experiment involving hint dissuasion.

We found that students with unusually high reading rates of hints were more likely to slow down from our dissuasion message "You asked for a hint quickly. Have you read the last hint?" though this data is only mildly statistically significant. We found that using a classifier of 3 words-per-second to tag hint gamers had a negative effect on students reading rate. This leads us to the conclusion that our choice was too aggressive and dissuaded engaged students from asking for hints. Real gamers failed to slow down below this rate because it is too tight. It is an easy ready rate to exceed.

Classifying and dissuading students with looser and more unusually fast reading rates had a better effect on reading rates. We were successful in dissuading students from reading hints too quickly when looser definitions of 6 words-per-second and 8 words-per-second were used as a gamer's reading rate.

These results are not as conclusive as we would like, though they show promise for future experiments. We'd like to rerun this experiment with different methodology to determine if we can find a stronger correlation between dissuasion and reading rates. We have discovered that a rate of 3 words-per-second is too aggressive and should no longer be used in our experiments; instead, we should perhaps look at values of 5, 6 and 8 words-per-second as the rate to classify gamers. This is still a work in progress. We will be conducting a more convincing experiment using all the experiment tools at our disposal in order to obtain more convincing results.

For the purpose of this research, we wished to test the user level experiments only and on that level, we believe that the framework is useful for evaluation of pedagogical design choices. The results of this experiment could have been stronger if used in conjunction with a curriculum level experiment where the students are given a pre and a post-test, and are given the same problem content. This is still a work in progress. We are will be

conducting a more convincing experiment using all the experiment tools at our disposal in order to obtain more convincing results.

I would like to thanks all the people associated with creating the Assistment system listed at www.assistment.org including the investigators Kenneth Koedinger, and Brian Junker at Carnegie Mellon. We would also like to acknowledge funding from the US Department of Education, the National Science Foundation, the Office of Naval Research and the Spencer Foundation.

8. References

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

[2] Anderson J.R., Pelletier, R., *A developmental system for model-tracing tutors*. In Lawrence Birnbaum, editor. The International Conference on the Learning Sciences. Association for the Advancement of Computing in Education. Pages 1-8, Charlottesville, Virginia, 1991

[3] Apache Axis http://ws.apache.org/axis/

[4] Baker, R.S., Corbett, A.T., Koedinger, K.R. (2004) *Detecting Student Misues of Intelligent Tutoring System.* Proceedings of the 7th International Conference on Intelligent Tutoring Systems, 531-540

[5] Cappel, D., Jewell, T. (2002) *Java Web Services*, O'Reilly & Associates, Sebastopol, CA

[6] Corbett, A. T., Koedinger, K. R., & Anderson, J. R. (1997). *Intelligent tutoring systems.* In Helander, M. G., Landauer, T. K., & Prabhu, P. V. (Ed.s) Handbook of Human-Computer Interaction, (pp. 849-874). Amsterdam, The Netherlands: Elsevier Science B. V.

[7] Eclipse Web Tools Platform http://www.eclipse.org/webtools/

[8] Feng, M., Heffernan, N.T., & Koedinger, K.R. (2006). *Addressing the Testing Challenge with a Web-Based E-Assessment System that Tutors as it Assesses*. Proceedings of the Fifteenth International World Wide Web Conference. pp. 307-316.

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

[10] Koedinger, K.R., Aleven, V., Heffernan. T., Mclaren, B. & Hockernberry, M. (2004) *Opening the Doors to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration*. Proceedings of 7th Annual Intelligent Tutoring Systems Conference, moaceio,, Brazil. Page 162-173

[11] Koedinger, K. R., & Anderson, J. R. (1998). *Illustrating principled design: The early evolution of a cognitive tutor for algebra symbolization*. Interactive Learning Environments, 5, 161-180.

[12] Kodaganallur, V. Weitz, R. Rosenthal, D. (2006) *Tools for Building Intelligent Tutoring System.* Proceedings of the 39th Hawaii International conference on System Sciences

[13] Murray, C. VanLehn, K. (2005) *Effects of Dissuading Unnecessary Help Requests While Providing Proacting Help*. Proceedings of the 12th International Conference on Artificial Intelligence In Education 887-889. Amsterdam: ISO Press.

[14] Mitrovic A, Ohlsson, S. *Evaluation of a constraint-based database language*. Internationl Journal of Artificial Intelligence in Eduation, 10(3-4):238-256, 1999

[15] Merrill, D., Reiser, J. (1994) *Scaffolding Effective Problem Solving Strategies in Interactive Learning Environments*. From the Proceedings of the 16th annual Conference of the Cognitive Science Society

[16] Murray, T., *Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art* International Journal of Artificial Intelligence in Education, 0, pp. 98-129

[17] Newcomer, E. Lomow, G. (2005) *Understanding SOA with Web Services* , Pearson Education, Inc. Upper Saddle River, NJ

[18] Nuzzo-Jones., G. Macasek M.A., Walonoski, J., Rasmussen K. P., Heffernan, N.T., *Common Tutor Object Platform, an e-Learning Software Development Strategy*. http://www.assistment.org/portal/project/papers/www/nuzzojones_etal.pdf

[19] Nuzzo-Jones, G., Walonoski, J.A., Heffernan, N.T., Livak, T. (2005). *The eXtensible Tutor Architecture: A New Foundation for ITS.* In C.K. Looi, G. McCalla, B. Bredeweg, & J. Breuker (Eds.) Proceedings of the 12th International Conference on Artificial Intelligence In Education, 902-904. Amsterdam: ISO Press.

[20] Razzaq, L., Feng, M., Nuzzo-Jones, G., Heffernan, N.T., Koedinger, K. R., Junker, B., Ritter, S., Knight, A., 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*. In C.K. Looi, G. McCalla, B. Bredeweg, & J. Breuker (Eds.) Proceedings of the 12th International Conference on Artificial Intelligence In Education, 555-562. Amsterdam: ISO Press.

[21] Razzaq, L., Heffernan, N.T. (2006). *Scaffolding vs. hints in the Assistment System*. In Ikeda, Ashley & Chan (Eds.). Proceedings of the 8th International Conference on Intelligent Tutoring Systems. Springer-Verlag: Berlin. pp. 635-644. 2006.

[22] Rickel, J., Ganeshan, R. (2000) *Task-Oriented Tutorial Dialogue: Issues and Agents*. In AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications, pages 52-67.

[23] TeachScheme, ReachJava! http://www.teach-scheme.org/

[24] Turner, T.E., Macasek, M.A., Nuzzo-Jones, G., Heffernan, N..T, Koedinger, K. (2005). *The Assistment Builder: A Rapid Development Tool for ITS*. In C.K. Looi, G. McCalla, B. Bredeweg, & J. Breuker (Eds.) Proceedings of the 12th Artificial Intelligence In Education, 929-931. Amsterdam: ISO Press.

[25] VanLehn, K. (2006) *The Behavior of Tutoring Systems*. International Journal of Artificial Intelligence in Education. 16, pages not determined yet.

[26] Walonoski, J., Heffernan, N.T. (2006). *Detection and Analysis of Off-Task Gaming Behavior in Intelligent Tutoring Systems.* In Ikeda, Ashley & Chan (Eds.). Proceedings of the 8th International Conference on Intelligent Tutoring Systems. Springer-Verlag: Berlin. pp. 382-391.

[27] Walonoski, J., Heffernan, N. T. (2006). *Prevention of Off-Task Gaming Behavior in Intelligent Tutoring Systems*. In Ikeda, Ashley & Chan (Eds.). Proceedings of the 8th International Conference on Intelligent Tutoring Systems. Springer-Verlag: Berlin. pp. 722-724.

[28] Young R.M. & Lewis, R.L., *The Soar Cognitive Architecture and Human Working Memory* (1999), Chapter 7 of: A. Miyake & P. Shah (eds), Models of Working Memory: Mechanisms of Active Maintenance and Executive Control, 224-256. Cambridge University Press.

APPENDIX I – JESS Addition Tutor

```
;; FOCUS-ON-FIRST-COLUMN
;; IF
;;    The goal is to do an addition problem
;;    And there is no pending subgoal (i.e., we've just started the problem)
;;    And C is the rightmost column of the table
;; THEN
;;    Set a subgoal to process column C
(defrule focus-on-first-column
;         (addition
;                   (problem ?problem))
          ?problem <- (problem
                    (subgoals )
                    (interface-elements $? ?table $?))
          ?table <- (table
                    (columns $? ?right-column))
          ?right-column <- (column
                    (cells $? ?first-addend ?second-addend ?result))
          ?first-addend <- (cell
                    (value ?num1))
          ?second-addend <- (cell
                    (value ?num2))
          ?result <- (cell
                    (value nil))
          ?special-tutor-fact <- (special-tutor-fact-correct)
=>
          (bind ?current-sub-goal (assert (process-column-goal
                    (column ?right-column)
                    (first-addend ?num1)
                    (second-addend ?num2))))
          (bind ?work-to-do (assert (finish-problem-goal
                    (begin-rule focus-on-first-column))))
          (modify ?problem
                    (subgoals (create$ ?current-sub-goal ?work-to-do)))
          (modify ?special-tutor-fact
                    (hint-message (construct-message [Start with the column
                                        on the right. This is the ones column ])))
;         (printout t "Focus-on-first-column." crlf)
)

;; FOCUS-ON-NEXT-COLUMN
;; IF
;;    The goal is to do an addition problem
;;    And there is no pending subgoal
;;    And C is the rightmost column with numbers to add and no result
;; THEN
```

```
;;    Set a subgoal to process column C

(defrule focus-on-next-column
;         (addition
;                   (problem ?problem))
          ?problem <- (problem
                    (subgoals ?work-to-do)
                    (interface-elements $? ?table $?))
          ?work-to-do <- (finish-problem-goal)
          ?table <- (table
                    (columns $? ?next-column ?previous-column $?))
          ?previous-column <- (column
                    (cells $? ?previous-result))
          ?previous-result <- (cell
                    (value ?val&:(neq ?val nil)))
          ?next-column <- (column
                    (name ?col-name)
                    (cells ?carry ?first-addend ?second-addend ?result)
                    (position ?pos))
          ?result <- (cell
                    (value nil))
          ?carry <- (cell
                    (value ?num0))
          ?first-addend <- (cell
                    (value ?num1))
          ?second-addend <- (cell
                    (value ?num2))
          ?special-tutor-fact <- (special-tutor-fact-correct)
=>
          (bind ?current-sub-goal (assert (process-column-goal
                    (column ?next-column)
                    (carry ?num0)
                    (first-addend ?num1)
                    (second-addend ?num2))))
          (modify ?work-to-do
                    (begin-rule focus-on-next-column))
          (modify ?problem
                              (subgoals (create$ ?current-sub-goal ?work-to-do)))
          (modify ?special-tutor-fact
                    (hint-message (construct-message [Move on to the ?pos column
                                                  from the right.This is the ?col-name
                                                  column.])))
          ;         (printout t "Focus-on-next-column." crlf)
)

;; ADD-ADDENDS
```

```
;; IF
;;    There is a goal to process column C
;; THEN
;;    Set Sum to the sum of the addends in column C
;;    And set a subgoal to write Sum as the result in column C
;;    And remove the goal to process column C

(defrule add-addends
;         (addition
;                   (problem ?problem))
          ?problem <- (problem
                    (subgoals $?sg1 ?subgoal $?sg2))
          ?subgoal <- (process-column-goal
                    (carry ?carry)
                    (first-addend ?num1&:(neq ?num1 nil))
                    (second-addend ?num2&:(neq ?num2 nil))
                    (column ?column)
                    (sum nil))
          ?special-tutor-fact <- (special-tutor-fact-correct)
=>
          (bind ?sum (+ ?num1 ?num2))
          (modify          ?subgoal
                    (sum ?sum))
          (modify ?special-tutor-fact
                    (hint-message (construct-message [You need to add the
                              two digits in this column. Adding ?num1 and ?num2
                              gives ?sum .])))
;         (printout t "Add addends." crlf)
)

;; ADD-CARRY
;; IF
;;    There is a goal to write Sum as the result in column C
;;    And there is a carry into column C
;;    And the carry has not been added to Sum
;; THEN
;;    Change the goal to write Sum+1 as the result
;;    And mark the carry as added

(defrule add-carry
;         (addition
;                   (problem ?problem))
          ?problem <- (problem
                    (subgoals $? ?subgoal $?))
          ?subgoal <- (process-column-goal
                    (sum ?sum&:(neq ?sum nil))
```

```
                (carry ?num0&:(neq ?num0 nil))
                (first-addend ?num1)
                (second-addend ?num2))
        (test (neq ?num0 nil))
        ?special-tutor-fact <- (special-tutor-fact-correct)
=>
        (bind ?new-sum (+ ?sum ?num0))
        (modify ?subgoal
                (sum ?new-sum)
                (carry nil))
        (modify ?special-tutor-fact
                (hint-message (construct-message [There is a carry in to
                        this column so you need to add the value carried
                        in. This gives ?sum + 1 equals ?new-sum .])))
;       (printout t "Add carry." crlf)
)

;; MUST-CARRY
;; IF
;;   There is a goal to write Sum as the result in column C
;;   And the carry into column C (if any) has been added to Sum
;;   And Sum > 9
;;   And Next is the column to the left of C
;; THEN
;;   Change the goal to write Sum-10 as the result in C
;;   Set a subgoal to write 1 as a carry in column Next

(defrule must-carry
;       (addition
;               (problem ?problem))
        ?problem <- (problem
                (interface-elements $? ?table $?)
                (subgoals $?sg1 ?subgoal $?sg2))
        ?subgoal <- (process-column-goal
                (sum ?sum&:(neq sum nil))
                (carry nil)
                (column ?column))
        (test (numberp ?sum))
        (test (> ?sum 9))
        ?column <- (column
                (name ?column-name))
        ?table <- (table
                (columns $? ?next-column ?column $?))
        ?next-column <- (column
                (position ?next-pos))
        ?special-tutor-fact <- (special-tutor-fact-correct)
```

```
=>
        (bind ?new-sum (- ?sum 10))
        (modify ?subgoal
                (sum ?new-sum))
        (bind ?write-carry-goal (assert (write-carry-goal
                (column ?next-column)
                (carry 1))))
        (modify ?problem
                (subgoals (create$ $?sg1 ?write-carry-goal ?subgoal $?sg2)))
        (modify ?special-tutor-fact
                (hint-message (construct-message [The sum that you have ?sum
                        is greater than 9. So you need to carry 10 of the
                        ?sum to the ?next-pos column. And you need to write
                        the rest of the sum at the bottom of the ?column-name
                        column.])))
;       (printout t "Must carry" crlf)
)

;; WRITE-SUM
;; IF
;;   There is a goal to write Sum as the result in column C
;;   And Sum < 10
;;   And the carry into column C (if any) has been added
;; THEN
;;   Write Sum as the result in column C
;;   And remove the goal

(defrule write-sum
;       (addition
;               (problem ?problem))
        ?problem <- (problem
                (subgoals $?sg1 ?subgoal $?sg2))
        ?subgoal <- (process-column-goal
                (sum ?sum&:(neq ?sum nil))
                (column ?column)
                (carry nil))
        (test (< ?sum 10))
        ?column <- (column
                (position ?pos)
                (cells $? ?result))
        ?result <- (cell
                        (name ?cell-name))
        ?special-tutor-fact <- (special-tutor-fact-correct)
=>
        (modify ?result
           (value ?sum))
```

```
        (modify ?problem
                (subgoals (create$ $?sg1 $?sg2)))
        (retract ?subgoal)
        (modify ?special-tutor-fact
                (selection ?cell-name)
                (action "UpdateTable")
                (input ?sum)
                (hint-message (construct-message [Write sum ?sum at the
                                bottom of the ?pos column.])))
;       (printout t "Write sum." crlf)
)

;; WRITE-CARRY
;; IF
;;   There is a goal to write a carry in column C
;;   And there is no result that has been recorded in the previous column
;;   And sum has been calculated in previous column P
;; THEN
;;   Write the carry in column C
;;   And remove the goal

(defrule write-carry
;       (addition
;               (problem ?problem))
        ?problem <- (problem
                (subgoals $?sg1 ?subgoal $?sg2)
                (interface-elements $? ?table $?))
        ?subgoal <- (write-carry-goal
                (carry ?num)
                (column ?column))
        ?column <- (column
                (position ?pos)
                (cells ?carry $?))
        ?carry <- (cell
                (name ?cell-name)
                (value nil))
        ?table <- (table
                (columns $? ?column ?previous-column $?))
        ?previous-column <- (column
                (position ?pos-previous)
            (cells $? ?sum))
      ?sum <- (cell
;               (value ?val&:(neq ?val nil))
                )
        ?special-tutor-fact <- (special-tutor-fact-correct)
=>
```

```
(printout t crlf "Selection: " ?cell-name "  Action: 'UpdateTable'  Input: " ?num crlf)
        (modify ?carry
                (value ?num))
        (modify ?problem
                (subgoals (create$ ?sg1 ?sg2)))         ; the remaining subgoals
        (modify ?special-tutor-fact
                (selection ?cell-name)
                (action "UpdateTable")
                (input ?num)
                (hint-message (construct-message [You need to complete
                                the work on the ?pos-previous column.]
                        [Write carry from the ?pos-previous
                                to the next column.]
                        [Write ?num at the top of the ?pos column
                                from the right.])))
        (retract ?subgoal)
;     (printout t "Write-carry." crlf)
)

;; BUGGY-FOCUS-ON-FIRST-COLUMN
;; IF
;;    The goal is to do an addition problem
;;    And there is no pending subgoal (i.e., we've just started the problem)
;;    And C is a column of the table but NOT the rightmost column
;; THEN
;;    Set a subgoal to process column C
;;    Set an error message "Start with the column all the way to the right, the ones column.
You've started in another column.

(defrule BUGGY-focus-on-first-column
;       (addition
;               (problem ?problem))
        (declare (salience -100))
        ?problem <- (problem
                (subgoals )
                (interface-elements $? ?table $?))
        ?table <- (table
                (columns $? ?wrong-column $? ?right-column))
        ?wrong-column <- (column
                (cells $? ?wrong-cell $?))
        ?right-column <- (column
                (cells $? ?first-addend ?second-addend ?result))
        ?wrong-cell <- (cell
           (name ?wrong-cell-name)
                (value nil))
        ?first-addend <- (cell
```

```
            (value ?num1))
        ?second-addend <- (cell
            (value ?num2))
        ?result <- (cell
           (name ?cell-name)
            (value nil))
        ?special-tutor-fact <- (special-tutor-fact-buggy)
=>
        (bind ?current-sub-goal (assert (process-column-goal
            (column ?right-column)
            (first-addend ?num1)
            (second-addend ?num2))))
        (modify ?problem
            (subgoals ?current-sub-goal))
        (modify ?special-tutor-fact
            (selection ?wrong-cell-name)
            (action DONT-CARE)
            (input DONT-CARE)
            (buggy-message (construct-message [Start with the column all the way to
the
                                right, the ones column.  You've started in another
column.])))
)

;; BUGGY-WRITE-CARRY
;; IF
;;   There is a goal to write a carry in column C
;;   And sum has not yet been calculated in previous column P
;; THEN
;;   Write the carry in column C
;;   And remove the goal

;(defrule Buggy-write-carry
;       (addition
;           (problem ?problem))
;       (declare (salience -100))
;       ?problem <- (problem
;           (subgoals $?sg1 ?subgoal $?sg2)
;           (interface-elements $? ?table $?))
;       ?subgoal <- (write-carry-goal
;           (carry ?num)
;           (column ?column))
;       ?column <- (column
;           (position ?pos)
;           (cells ?carry $?))
;       ?carry <- (cell
```

```
;                   (name ?cell-name)
;                   (value nil))
;         ?table <- (table
;                   (columns $? ?column ?previous-column $?))
;         ?previous-column <- (column
;                   (position ?pos-previous)
;                (cells $? ?sum))
;      ?sum <- (cell
;                   (value ?val&:(eq ?val nil)))
;         ?special-tutor-fact <- (special-tutor-fact-buggy)
;=>
;         (modify ?carry
;                   (value ?num))
;         (modify ?problem
;                   (subgoals ?sg1 ?sg2)) ; the remaining subgoals
;         (modify ?special-tutor-fact
;                   (selection ?cell-name)
;                   (action "UpdateTable")
;                   (input ?num)
;                   (buggy-message (construct-message [You need to write the
;                                       sum before doing the carry.])))
;         (retract ?subgoal)
;)

;(defrule Done
;         ?problem <- (problem
;                   (subgoals ?subgoal)
;                   (interface-elements $? ?table $?))
;         ?subgoal <- (finish-problem-goal
;                   (begin-rule ?beginning-rule))
;         ?table <- (table
;                   (columns $? ?a-column))
;         ?a-column <- (column
;                   (cells $? ?first-addend ?second-addend ?result))
;         ?first-addend <- (cell
;                   (value ?num1&:(neq ?num1 nil)))
;         ?second-addend <- (cell
;                   (value ?num2&:(neq ?num2 nil)))
;         ?result <- (cell
;                (name ?cell-name)
;                   (value ?num3&:(neq ?num3 nil)))
;         ?special-tutor-fact <- (special-tutor-fact-correct)
;=>
;         (printout t crlf "Problem done; began with rule " ?beginning-rule crlf)
;         (modify ?special-tutor-fact
;                   (selection done)
```

```
;               (action ButtonPressed)
;               (input -1)
;               (hint-message (construct-message [ Click on the Done ;Button. ] )))
;)
```

APPENDIX II – Hint Experiment Results

#	STUDENT	DUSSUADED	ReadingRate1WPS	RateWordPerSecond
1	21971	TRUE	12.81337	0.220216
2	22982	TRUE	14.22222	3.2
3	24699	TRUE	6.42978	9.940358
4	26982	FALSE	18.2704	27.64977
5	27756	TRUE	6.930289	6.021409
6	31340	TRUE	13.2626	10.39501
7	31689	FALSE	7.943925	9.742519
8	31693	FALSE	6.605691	1.180538
9	31695	FALSE	3.263052	26.49007
10	31703	TRUE	6.622517	5.032022
11	31714	FALSE	25.15244	1.178711
12	31741	FALSE	21.33333	16
13	31748	FALSE	23.28289	18.70079
14	31762	TRUE	4.74193	0.577717
15	31765	FALSE	7.735584	2.942258
16	31767	TRUE	3.14095	5.91716
17	31770	FALSE	12.70151	8
18	31783	TRUE	7.246377	13.33333
19	31801	FALSE	4.385965	3.228647
20	31810	FALSE	16.26898	2.434077
21	31825	FALSE	4.074784	5.611672
22	31835	TRUE	10.26856	0.75
23	31862	FALSE	5.452821	1.463823
24	31887	FALSE	6.5	2.448409
25	31895	FALSE	3.199606	10.27668
26	31907	TRUE	6.178288	0.787557
27	31911	TRUE	4.962779	5.235602
28	31942	FALSE	4.062976	1.656658
29	31958	FALSE	5.33224	2.881251
30	31969	TRUE	3.294118	1.36854
31	31974	TRUE	12.26667	6.900878
32	31981	FALSE	20.55197	3.9783
33	31986	TRUE	6.402049	14.04056
34	32001	TRUE	21.33333	16.34877
35	32010	TRUE	5.118362	9.017133
36	32014	TRUE	4.706199	1.50922
37	32016	FALSE	10.88348	4.415011
38	32018	FALSE	13.41589	13.76421
39	32020	FALSE	3.271028	0.042414
40	32022	FALSE	22.35772	3.670687
41	32023	FALSE	5.317054	5.646173
42	32058	TRUE	3.412969	2.82657
43	32059	FALSE	9.299781	1.416932

44	32065	TRUE	5.014749	10.29748
45	32066	TRUE	13.37154	1.71809
46	32073	TRUE	14.49275	4.889976
47	32076	FALSE	3.827228	2.031635
48	32078	TRUE	3.594536	0.927321
49	32095	TRUE	7.556675	3.434951
50	32128	FALSE	11.55556	0.755144
51	32134	TRUE	3.428571	0.156826
52	32137	FALSE	19	1.635497
53	32153	FALSE	4.150284	1.212268
54	32158	TRUE	8.633824	0.023751
55	32172	TRUE	3.748558	14.21801
56	32178	TRUE	5.538462	4.649499
57	32183	FALSE	3.575259	6.993007
58	32184	TRUE	8.347245	3.615329
59	32199	TRUE	4.583715	1.081617
60	32213	FALSE	8.858268	21.65354
61	32223	FALSE	3.067485	1.806942
62	32255	TRUE	7.705193	2.885624
63	32260	FALSE	17.05757	5.847953
64	32275	TRUE	5.271084	1.222179
65	32281	TRUE	3.764706	4.547044
66	32311	TRUE	18.89535	15.05646
67	32321	FALSE	3.699593	11.15023
68	32324	TRUE	13.07639	11.29235
69	32326	FALSE	12	8
70	32332	FALSE	16.85393	7.049892
71	32335	FALSE	4.205607	6.956522
72	32342	FALSE	4.697592	27.86885
73	32352	FALSE	9.918846	28.07018
74	32353	FALSE	8.726568	0.311883
75	32358	TRUE	22.22222	5.858495
76	32385	FALSE	8.830022	19.68827
77	32392	FALSE	4.096262	2.540835
78	32396	FALSE	5.445882	2.526316
79	32405	TRUE	4.055881	8.758141
80	32413	FALSE	8.87199	8.411215
81	32419	FALSE	4.740658	3.534884
82	32443	TRUE	18.66667	4.922644
83	32445	FALSE	14.36969	1.457938
84	32456	FALSE	4.210526	3.333333
85	32466	TRUE	9.040334	3.488019
86	32474	FALSE	4.724409	0.619323
87	32480	FALSE	31.14187	65.97938
88	32507	FALSE	9.057971	8.626887
89	32645	TRUE	3.076923	5.225343

90	32655	TRUE	4.991192	2.091175
91	32656	FALSE	3.333333	12.26158
92	32663	TRUE	4.666667	5.938634
93	32686	TRUE	3.11311	0.355037
94	32721	TRUE	12.5638	3.702724
95	32724	FALSE	27.35711	7.143899
96	32729	TRUE	5.003882	50.55292
97	32730	FALSE	5.707297	1.438159
98	32732	TRUE	9.501188	0.135651
99	32733	TRUE	14.15246	24.98048
100	32734	TRUE	4.923255	0.875226
101	32738	TRUE	5.025126	0.750751
102	32739	FALSE	5.230769	0.588192
103	32741	TRUE	3.725025	1.896409
104	32742	FALSE	4.934845	13.60202
105	32744	FALSE	5.293551	0.559284
106	32747	FALSE	6.121313	0.681558
107	32748	FALSE	4.413793	1.578947
108	32749	FALSE	4.237288	3.512195
109	32750	TRUE	20.86231	1.858736
110	32751	TRUE	4.18556	0.958641
111	32754	FALSE	4.232804	0.655503
112	32758	TRUE	11.46679	4.197272
113	32759	TRUE	3.236246	1.189189
114	32761	FALSE	3.886113	6.330992
115	32763	FALSE	5.293673	1.09529
116	32766	FALSE	9.020619	2.5
117	32767	FALSE	4.07767	36.45833
118	32769	TRUE	5.214506	0.151748
119	32861	TRUE	11.7712	3.103181
120	33153	FALSE	5.375381	3.515521
121	33155	TRUE	3.005658	0.417116
122	33163	FALSE	7.92752	7.824726
123	33701	TRUE	3.272727	7.616146
124	33716	TRUE	3.571429	3.502627
125	33782	TRUE	6.966434	0.339664
126	34885	FALSE	5.733945	5.204719
127	34886	TRUE	9.864365	4.263256
128	34889	FALSE	3.557568	5.005776
129	35025	FALSE	3.368421	3.104519
130	35624	TRUE	3.425275	1.66579
131	35661	TRUE	6.157965	4.725086
132	36004	TRUE	6.650042	23.13167
133	36009	FALSE	18.7638	8.858268
134	36092	FALSE	16.15911	0.725426
135	36094	FALSE	21.33195	15.84022

136	36095	FALSE	10.82056	21.34831
137	36096	TRUE	4.518072	13.46801
138	36097	FALSE	5.544355	0.84246
139	36101	TRUE	19	4.653869
140	36104	TRUE	4.978663	13.02378
141	36105	TRUE	10.03764	8.891523
142	36107	TRUE	10.66983	2.012477
143	36247	TRUE	10.67169	19.89619
144	36271	TRUE	3.815176	24.19984
145	36399	TRUE	4.592423	4.222081
146	36404	TRUE	4.339964	3.871681
147	36405	TRUE	3.995108	12.96071
148	36408	TRUE	5.33049	9.6

APPENDIX III – Web Service Interface (WSDL)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://stateservice.webservices.assistment.org"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://stateservice.webservices.assistment.org"
xmlns:intf="http://stateservice.webservices.assistment.org"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.3
Built on Oct 05, 2005 (05:23:37 EDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified"
targetNamespace="http://stateservice.webservices.assistment.org"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="getStateName">
    <complexType>
     <sequence>
      <element name="id" type="xsd:long"/>
      <element name="action" type="impl:Action"/>
     </sequence>
    </complexType>
   </element>
   <complexType name="StrategyProperty">
    <sequence>
     <element name="name" nillable="true" type="xsd:string"/>
     <element name="value" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="ArrayOfStrategyProperty">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item"
type="impl:StrategyProperty"/>
    </sequence>
   </complexType>
   <complexType name="Action">
    <sequence>
     <element name="elementID" nillable="true"
type="xsd:string"/>
     <element name="properties" nillable="true"
type="impl:ArrayOfStrategyProperty"/>
```

```
      <element name="type" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <element name="getStateNameResponse">
     <complexType>
      <sequence>
       <element name="getStateNameReturn" type="xsd:string"/>
      </sequence>
     </complexType>
    </element>
    <element name="getStateStrategy">
     <complexType>
      <sequence>
       <element name="id" type="xsd:long"/>
       <element name="action" type="impl:Action"/>
      </sequence>
     </complexType>
    </element>
    <element name="getStateStrategyResponse">
     <complexType>
      <sequence>
       <element name="getStateStrategyReturn"
type="impl:Strategy"/>
      </sequence>
     </complexType>
    </element>
    <complexType name="StrategyChild">
     <sequence>
      <element name="data" nillable="true" type="xsd:string"/>
      <element name="index" type="xsd:int"/>
      <element name="type" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <complexType name="ArrayOfStrategyChild">
     <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="item"
type="impl:StrategyChild"/>
     </sequence>
    </complexType>
    <complexType name="Strategy">
     <sequence>
      <element name="children" nillable="true"
type="impl:ArrayOfStrategyChild"/>
      <element name="properties" nillable="true"
```

```
type="impl:ArrayOfStrategyProperty"/>
      <element name="type" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <element name="getDefaultStrategy">
     <complexType>
      <sequence>
       <element name="id" type="xsd:long"/>
      </sequence>
     </complexType>
    </element>
    <element name="getDefaultStrategyResponse">
     <complexType>
      <sequence>
       <element name="getDefaultStrategyReturn"
type="impl:Strategy"/>
      </sequence>
     </complexType>
    </element>
    <element name="getHintStrategy">
     <complexType>
      <sequence>
       <element name="id" type="xsd:long"/>
      </sequence>
     </complexType>
    </element>
    <element name="getHintStrategyResponse">
     <complexType>
      <sequence>
       <element name="getHintStrategyReturn"
type="impl:Strategy"/>
      </sequence>
     </complexType>
    </element>
    <element name="emptyState">
     <complexType>
      <sequence>
       <element name="id" type="xsd:long"/>
      </sequence>
     </complexType>
    </element>
    <element name="emptyStateResponse">
     <complexType>
      <sequence>
```

```
      <element name="emptyStateReturn"  type="xsd:boolean"/>
     </sequence>
    </complexType>
   </element>
  </schema>
 </wsdl:types>
   <wsdl:message name="getStateNameResponse">
      <wsdl:part element="impl:getStateNameResponse"
name="parameters"/>
   </wsdl:message>
   <wsdl:message name="getHintStrategyRequest">
      <wsdl:part element="impl:getHintStrategy"
name="parameters"/>
   </wsdl:message>
   <wsdl:message name="emptyStateRequest">
      <wsdl:part element="impl:emptyState"  name="parameters"/>
   </wsdl:message>
   <wsdl:message name="emptyStateResponse">
      <wsdl:part element="impl:emptyStateResponse"
name="parameters"/>
   </wsdl:message>
   <wsdl:message name="getHintStrategyResponse">
      <wsdl:part element="impl:getHintStrategyResponse"
name="parameters"/>
   </wsdl:message>
   <wsdl:message name="getStateNameRequest">
      <wsdl:part element="impl:getStateName"  name="parameters"/>
   </wsdl:message>
   <wsdl:message name="getDefaultStrategyRequest">
      <wsdl:part element="impl:getDefaultStrategy"
name="parameters"/>
   </wsdl:message>
   <wsdl:message name="getDefaultStrategyResponse">
      <wsdl:part element="impl:getDefaultStrategyResponse"
name="parameters"/>
   </wsdl:message>
   <wsdl:message name="getStateStrategyResponse">
      <wsdl:part element="impl:getStateStrategyResponse"
name="parameters"/>
   </wsdl:message>
   <wsdl:message name="getStateStrategyRequest">
      <wsdl:part element="impl:getStateStrategy"
name="parameters"/>
   </wsdl:message>
```

```
   <wsdl:portType name="StateBehaviorService">
      <wsdl:operation name="getStateName">
         <wsdl:input message="impl:getStateNameRequest"
name="getStateNameRequest"/>
         <wsdl:output message="impl:getStateNameResponse"
name="getStateNameResponse"/>
      </wsdl:operation>
      <wsdl:operation name="getStateStrategy">
         <wsdl:input message="impl:getStateStrategyRequest"
name="getStateStrategyRequest"/>
         <wsdl:output message="impl:getStateStrategyResponse"
name="getStateStrategyResponse"/>
      </wsdl:operation>
      <wsdl:operation name="getDefaultStrategy">
         <wsdl:input message="impl:getDefaultStrategyRequest"
name="getDefaultStrategyRequest"/>
         <wsdl:output message="impl:getDefaultStrategyResponse"
name="getDefaultStrategyResponse"/>
      </wsdl:operation>
      <wsdl:operation name="getHintStrategy">
         <wsdl:input message="impl:getHintStrategyRequest"
name="getHintStrategyRequest"/>
         <wsdl:output message="impl:getHintStrategyResponse"
name="getHintStrategyResponse"/>
      </wsdl:operation>
      <wsdl:operation name="emptyState">
         <wsdl:input message="impl:emptyStateRequest"
name="emptyStateRequest"/>
         <wsdl:output message="impl:emptyStateResponse"
name="emptyStateResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="StateBehaviorServiceSoapBinding"
type="impl:StateBehaviorService">
      <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getStateName">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getStateNameRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="getStateNameResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
```

```
      </wsdl:operation>
<wsdl:operation name="getStateStrategy">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getStateStrategyRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="getStateStrategyResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getDefaultStrategy">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getDefaultStrategyRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="getDefaultStrategyResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getHintStrategy">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getHintStrategyRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="getHintStrategyResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="emptyState">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="emptyStateRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="emptyStateResponse">
            <wsdlsoap:body use="literal"/
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="StateBehaviorServiceService">
      <wsdl:port binding="impl:StateBehaviorServiceSoapBinding"
name="StateBehaviorService">
         <wsdlsoap:address
location="http://nth6.wpi.edu/StateBehaviorService/services/State
```

```
BehaviorService"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```