The Common Tutor Object Platform

By

Goss Nuzzo-Jones

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

In

Computer Science

by

____

December 2005

APPROVED:

____

Professor Neil T. Heffernan, Thesis Advisor

____

Professor Murali Mani, Reader

____

Professor Michael Gennert, Head of Department

ACKNOWLEDGEMENTS

I would also like to acknowledge my advisor, Dr. Neil Heffernan, for years of support and guidance. Also, thank you to Dr. Murali Mani for offering his feedback and supervision as reader of this thesis. Thank you to the Assistments Lab team, for their hard work in realizing these ideas, as well as their camaraderie. Finally, I would like to acknowledge the US Department of Education, National Science Foundation, Office of Naval Research, and the Worcester Public School System for providing the means to accomplish this research.

1. ABSTRACT

The Common Tutor Object Platform (CTOP) was designed as a lightweight component framework for creating and deploying applications relating to Intelligent Tutoring Systems. The CTOP supports a runtime for intelligent tutoring system content deployment, a content development environment, an extensive reporting tool, and other smaller applications. The CTOP was designed with future development in mind, allowing easy specification of new base objects and extension points for future development. It has been used as the foundation of the Assistments Project, a wide scale server based ITS deployment. This thesis documents the software engineering aspects of the project. The Assistments Project is capable of supporting a quarter of targeted students in Massachusetts, and optimistically scalable to the entire state and beyond.

2. Table of Contents

3. Table of Figures

4. INTRODUCTION

Intelligent Tutoring Systems (ITS) have been demonstrated in the past to be an effective means of educating students [8]. However many of the ITS strong points are eclipsed by the high cost involved in the cost of construction of the system. The Office of Naval Research has funded the Assistments Project to develop tools which reduce the cost of development of ITS. It has been estimated that for one hour of content that is delivered via an ITS it requires upwards of 200 hours of content development time [13, 1]. In order to produce content the author needs to be highly knowledgeable in several areas including the writing of complex production rules that requires a cognitive science background. Generally speaking most users and potential content developers do not have the sophisticated background required to adequately develop content for an ITS. Many systems have attempted to lower the content development time and recently the Assistment Project has been able to significantly reduce the time by limiting the complexity of the content that can be developed [18].

The term Intelligent Tutoring Systems covers a wide range of possible computer-based tutors, from cognitive model tracing tutors [3], constraint-based tutors [11], 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 [2, 10]. Student actions trigger transitions in the graph, and the current state of the problem is stored by the graph. Pseudo-tutors have nearly identical behavior to a rule-based tutor, but suffer from having no ability to generalize to different problems [3]. This pseudo-tutor approach allows for predicted behaviors and provides feedback based on those behaviors.

In this thesis there will be a focus on the Assistment Project, but there are many other ITS systems available. The Cognitive Tutor Authoring Tools [10] developed at Carnegie Mellon University offer a robust system devoted to work space tutors. The Online Learning Initiative (OLI) [15], also from Carnegie Mellon University, offers tutors on many subjects and is distributed over the Internet. The National Center for Research on Evaluation, Standards and Student Testing (CRESST) [19] offers a suite of online tools to develop content, however this ITS is limited in that the questions are open ended and require human intervention for assessing the answers.

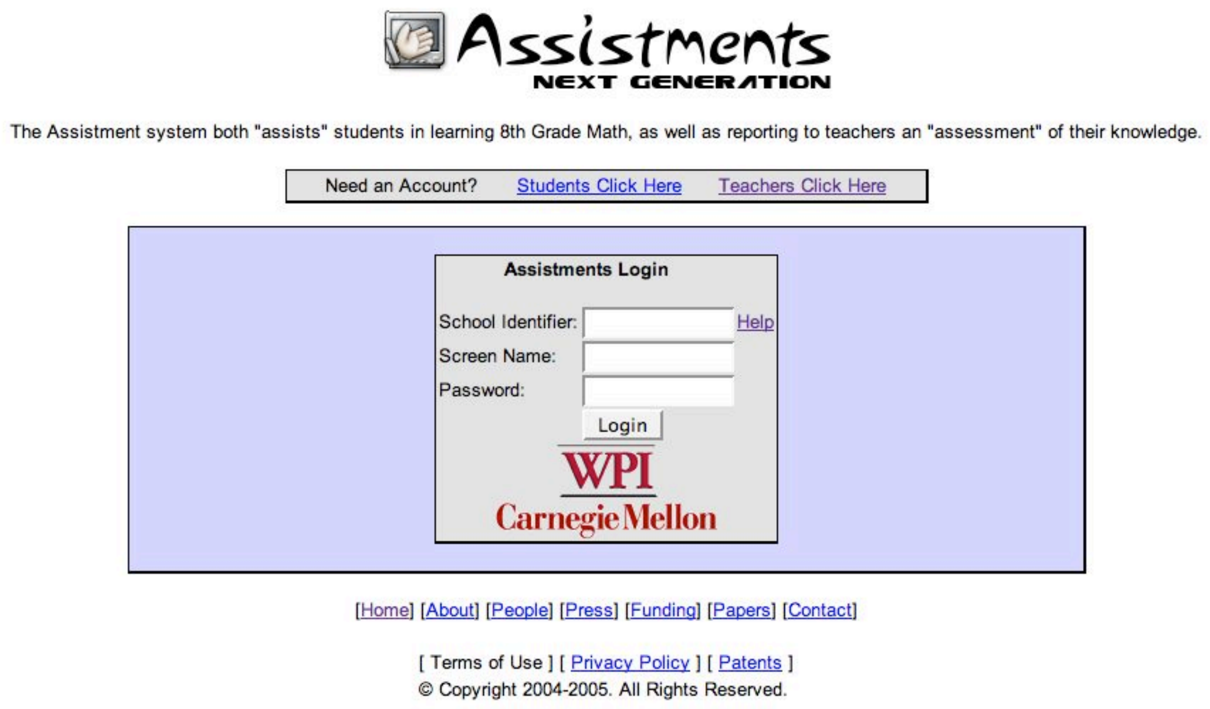
The success of ITS in general is well known, demonstrating useful learning effects [10]. There have been ITS deployed on a wide scale [10], but they suffered from some limitations, such as a lack of centralized logging, upgrade difficulties, and tutor strategy inflexibility. It has been shown that centralized logging of student actions in databases for experimental analysis is valuable [12]. This work sought to address these issues, as well as provide a rich feature base for future development of all tutor types.

The Assistment Project was previously built on top of the eXtensible Tutor Architecture (XTA) [14] which easily allowed for the extendibility of the system to increase functionality. When developed the XTA proved to be a reliable system however as time passed many of the faults of the XTA became evident the biggest of which was scalability. This prompted the Assistment Project Team to reevaluate the XTA and devise a new architecture that embodied many of the same principles of the XTA but also solved many of the ongoing issues present in the XTA. Out of this redesign the Common Tutor Object Platform (CTOP) was created. This new architecture is the subject of this thesis.

4.1 Assistments Project

The Assistments Project [16] is a multi-pronged educational software project (see Figure 1) with three primary goals. The first goal is to provide intelligent tutoring system content to students in a platform independent manner. The second goal is to provide the teachers of those students with fine-grained, useful reports identifying the strengths and weaknesses of those students. Finally, the third goal is providing a rapid development tool for creating intelligent tutoring system content.

Over the past year, the system has undergone development to provide core functionality to our first target audience, students preparing for the MCAS test in 8th grade. This academic year, tutoring content will be provided to 10th grade students in Massachusetts.



The Assistment system both "assists" students in learning 8th Grade Math, as well as reporting to teachers an "assessment" of their knowledge.

Need an Account? Students Click Here Teachers Click Here

Assistments Login

School Identifier: Help
Screen Name:
Password:
Login

WPI
Carnegie Mellon

[Home] [About] [People] [Press] [Funding] [Papers] [Contact]

[Terms of Use] [Privacy Policy] [Patents]
© Copyright 2004-2005. All Rights Reserved.

Figure 1 - Assistments Homepage

4.1.1 Goal of CTOP

The goal of this thesis was to create a component framework and API for developing applications dealing with Intelligent Tutoring Systems. This framework grew from the runtime XTA described in [14], as well as providing support for other applications. In this document, I will first examine the architecture of CTOP, then move into specific application instantiations, and conclude with anecdotal and scalability results from those applications and their development.

5. ARCHITECTURE

The CTOP is not a full feature component model (i.e. Enterprise Java Beans or .NET Framework), as such a replication of existing technology would be redundant and expensive. However, CTOP provides some services and features similar to existing component models, allowing developers to engineer their component-based applications on top of this platform.

5.1 Core Object Model

The core object model consists of a series of components considered to be universally applicable in many different pieces of ITS software. These core objects focus on content management and representation, as well as complex metadata associated with that content.

Content is rooted in *curriculum* components, which represent a series of *problems*. *Curriculums* are constructed prior to runtime by an outside author. The *curriculum* unit can be conceptually subdivided into two main pieces: the *curriculum* and *sections*. The *curriculum* is composed of one or more *sections*, with each *section* containing *problems* or other *sections.* This recursive structure allows for a rich hierarchy of different types of *sections* and *problems*.

The *section* sub-component is an abstraction for a particular listing of problems. This abstraction has been extended to implement our current *section* types, and allows for future expansion of the *curriculum* unit. Currently existing *section* types include "Linear" (*problems* or sub-*sections* are presented in linear order), "Random" (*problems* or sub-*sections* are presented in a pseudo-random order), and "Experiment" (a single *problem* or sub-*section* is selected pseudo-randomly from a list, the others are ignored). The *progress* saves an individual student's state about a given shared *curriculum* and its *sections*. Also contained within the *progress* is metadata such as total number of *problems* completed and the last updated time.

The *problem* component represents a problem to be tutored, including questions and answers required to solve the problem. Each of these questions is represented by a *problem* composed of two main pieces: an *interface* and a *behavior*.

The *interface* definition is interpreted by the runtime and displayed for viewing and interaction to the user. This display follows a two-step process, allowing for easy customization of platform and interface specification. The *interface* definition consists of "high-level" *interface* elements ("widgets"), which can have complex behavior (multimedia, spell-checking text fields, algebra parsing text fields). These "high-level" widgets have a representation in the runtime composed of "low-level" widgets. "Low-level" widgets are widgets common to many possible platforms of *interface*, and include text labels, text fields, images, radio buttons, etc.

The *behaviors* for each *problem* define the results of actions on the *interface*. An action might consist of pushing a button or selecting a radio button. Examples of *behavior* definitions are state graphs, cognitive model tracing, or constraint tutoring, defining the interaction that a specific *interface* definition possesses. Several types of behaviors presently exist (state graph tutor, JESS cognitive model), but the interpretation and programmatic response to the behaviors is up to the consuming application, such as the runtime described below.

Behaviors interact with applications built on the CTOP by producing and consuming *actions*. These *actions* are representations of state changes in a specific problem *interface*. The CTOP provides definitions of generic *actions*, as well as *actions* for each type of *interface* widget. These *actions* form a messaging layer that allows for communication between components. To facilitate scalability and loose coupling of components, these *actions* are XML based and can be passed over a network connection.

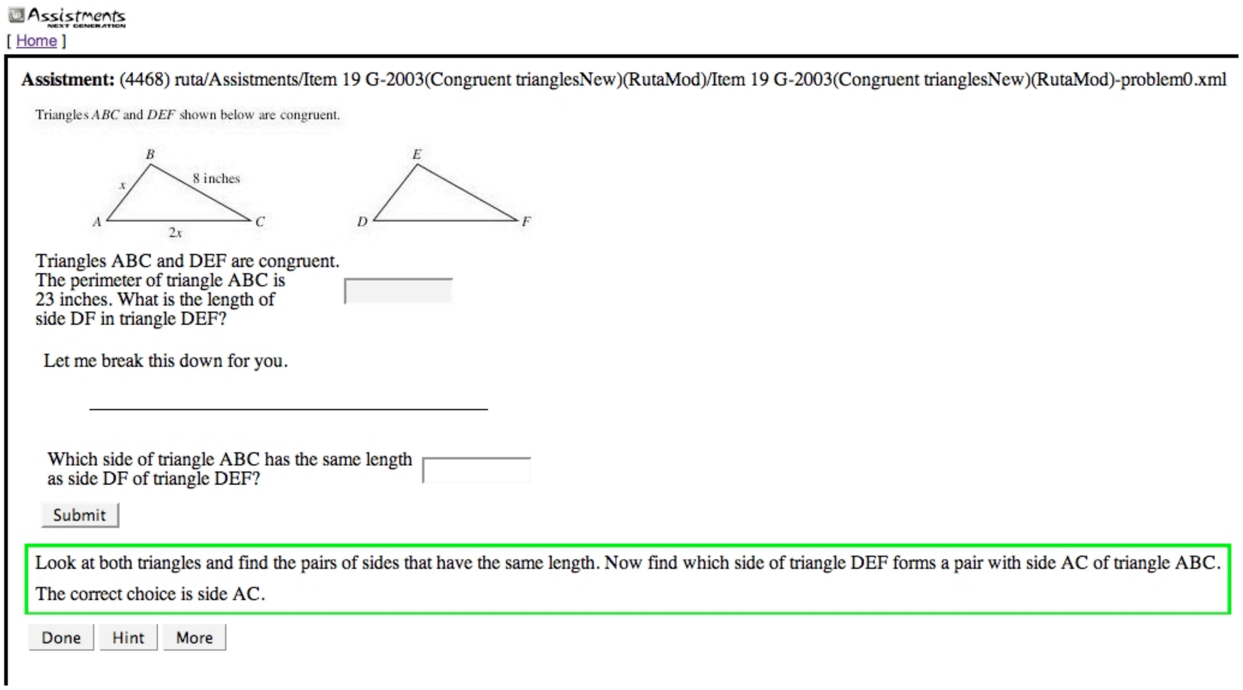
Transfer models are another component provided to consuming applications, and these components are used for concept mapping. *Transfer models* provide a metadata store of a network of *problems* related to *knowledge components*. This mapping provides a way to track student knowledge over time, as well as a way to organize *problems* in a hierarchal fashion with regard to the content of the *problem. Transfer models* can be used to provide a rich model of student knowledge, as well as a metric for comparing the value of different problem organizational structures.

Finally, there are generic component types, which can be associated with virtually every other component in CTOP. These include *properties* and *preferences*, which provide metadata, both time and user specific about specific components or instantiated objects.

5.2 Datalayer

The Datalayer's function is to decouple the applications built on CTOP from storing and retrieving content objects. Previous implementations of the Assistment system had embedded file system calls, as well as SQL statements buried within the application code. Objects contained knowledge of how they were to be stored and in what format. In the move towards the component-based architecture, it was decided to divorce objects from this knowledge of persistence. The philosophy of the Datalayer is that objects should not directly know how to persist themselves, but instead have access to all data that needs to be persisted.

The Datalayer also provides a level of transparency to the CTOP. Users of the CTOP easily access our core objects through the simple Datalayer API, and never worry about storage mechanisms. This allows for different Datalayers that all follow the same API to be easily swapped and CTOP applications can remain unaware. In fact, multiple data sources can be used at the same time, allowing different types of components to be stored in different mediums simultaneously. For instance, it may be beneficial for some components to be serialized to a relational database, whereas others would be more effectively stored on a file system.



Assistment: (4468) ruta/Assistments/Item 19 G-2003(Congruent trianglesNew)(RutaMod)/Item 19 G-2003(Congruent trianglesNew)(RutaMod)-problem0.xml

Triangles *ABC* and *DEF* shown below are congruent.

Triangles ABC and DEF are congruent. The perimeter of triangle ABC is 23 inches. What is the length of side DF in triangle DEF?

Let me break this down for you.

Which side of triangle ABC has the same length as side DF of triangle DEF?

Look at both triangles and find the pairs of sides that have the same length. Now find which side of triangle DEF forms a pair with side AC of triangle ABC.

The correct choice is side AC.

Figure 2 - Runtime

Each component's interface contains methods that provide access to the object's persisted data. These persistence methods are shared for every instance of that component. For example, every *behavior* component persists a unique ID, a type, a description, and a link to an interface. The Datalayer uses these methods to create some storable media. The current implementation creates an XML file that represents the object, and then is stored in the database. It is quite conceivable that this file could also be stored directly onto a file system, or sent across the network to another machine. A previous implementation of the Datalayer used relational persistence to store the object structure to a relational database. It did this using the tool Hibernate [7], which was later replaced due to scalability issues.

5.3 Extensibility

The CTOP was designed with extensibility in mind. All of the components described above provide interfaces for their interaction and can thus be easily overridden by a developer. There are also obvious points of coupling where other providers can easily be swapped in and out, such as in the Datalayer, using a variety of methods for persistence.

CTOP provides a number of API's to handle some lifecycle functions, as well as interaction with various components. The Datalayer described above provides an API that creates inflated components of the various types to a consuming application. This API also handles interaction with various component metadata stores. A separate API is provided for interaction with *transfer models*, as these components have a more complex function than others in the framework.

An additional API is created by the events generated by *problems* as *actions*. The *actions* are generated by individual interface components and thus are not located in a single entity; however, they follow a standard format and can be viewed as an XML service of sorts.

6. APPLICATIONS

There are a number of applications that presently make up the Assistments project, and a number of additional applications and extensions in development. All of these reuse code from the CTOP, some more than others. The most mature and complete pieces of software are detailed here.

6.1 Runtime

The runtime application (see Figure 2) existed previous to the creation of CTOP, as the eXtensible Tutor Architecture (XTA) [14]. However, with the creation of CTOP, the runtime was developed to be more modular, allowing it to interact easily with other applications. The runtime serves as a content deployment application, the primary means of content delivery for the Assistments Project. Its purpose is to guide a student through a *curriculum* that consisting of *problems*. The CTOP objects comprise the majority of the runtime behavior. Upon access of the runtime, the *curriculum* and the student's *progress* must be retrieved from the Datalayer. The runtime retrieves the current *problem* from the *curriculum* and outputs it to the student. After a student has performed one or more *actions*, the runtime reacts to those *actions* and updates the *problem* and *interface* appropriately. In this sense the runtime also acts as an event handler for the core component, translating actions from the user to the objects, and representing this in its output.

There is also a set of important componentized objects that the runtime relies on. The *agenda* controls the ordering of problems outside of the *curriculum* at execution time. Problems contain *strategies* that can alter the *agenda* dynamically. This *agenda* provides an innovative dynamic staging of problems. The *agenda* is an instantiated *curriculum*, and its structure is initially based on the contents of that *curriculum*. However, this structure can be changed, as indicated above, by s*trategies* at execution time. Also, the *agenda* provides references to the instantiated *problems* within it, allowing routing of *actions* and other events. There is also a logging component with in the runtime that records every student action taken and responded to. This is useful for the assessment of students, providing detailed reports to teachers. It is also used to detect student "off-task behavior" and to replay through problems step-by-step if a student reattempts an unfinished problem. This has provided a high-level view of the runtime's functionality, now let us delve deeper.

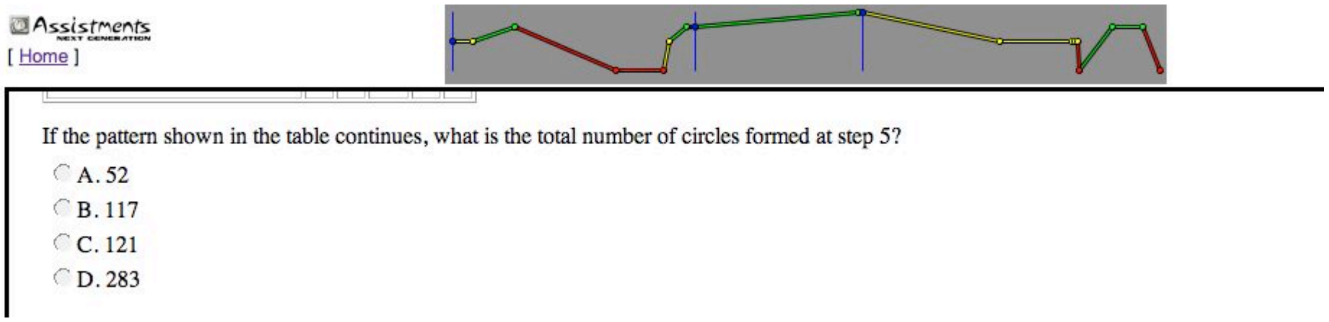
6.1.1 Runtime Architecture

As mentioned above, the *agenda* is a critical element of the runtime application. Contained within the agenda is a ordering of problems and tutoring messages (hints or bug messages). The contents of the *agenda* are operated upon by the various *tutor strategies*, selecting new *problems* from *sections* (possibly within *sections*) within a *curriculum* to append and choosing the next *problem* to present. The *agenda* in conjunction with *tutor strategies* allows for high-level control over *problems* and provides flow control between *problems.* For instance, a scaffolding *tutor strategy* arranges a number of *problems* in a tree structure, or scaffold. When the student answers the root *problem* incorrectly, a sequence of other *problems* associated with that incorrect answer is queued for presentation to the student. These scaffolding *problems* can continue to branch as the roots of their own tree.

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

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

Capturing these events has given an assortment of data to analyze for a variety of needs. User action data captured allows for examination of usage-patterns, including detection of system gaming (superficially going through tutoring-content without actually trying to learn) [20]. This data also enables us to quickly build reports for teachers on their students, as well as giving a complete trace of student work. This trace allows us to replay a user's session, which could be useful for quickly spotting fundamental misunderstandings on the part of the user, as well as debugging the content and the system itself (by attempting to duplicate errors).



If the pattern shown in the table continues, what is the total number of circles formed at step 5?

- A. 52
- B. 117
- C. 121
- D. 283

Figure 3 - Visual Feedback on Student Actions

An emerging role of the runtime is to perform instructional method comparisons. This is a new research topic for our system. Early experiments use student log data in order to detect gaming behavior, such as quickly exhausting hints for questions without giving an attempt at the problem. A result of this is a new method of providing visual representation of a students gaming index on the screen, to give visual cues to instructors to intervene (see Figure 3) [20].

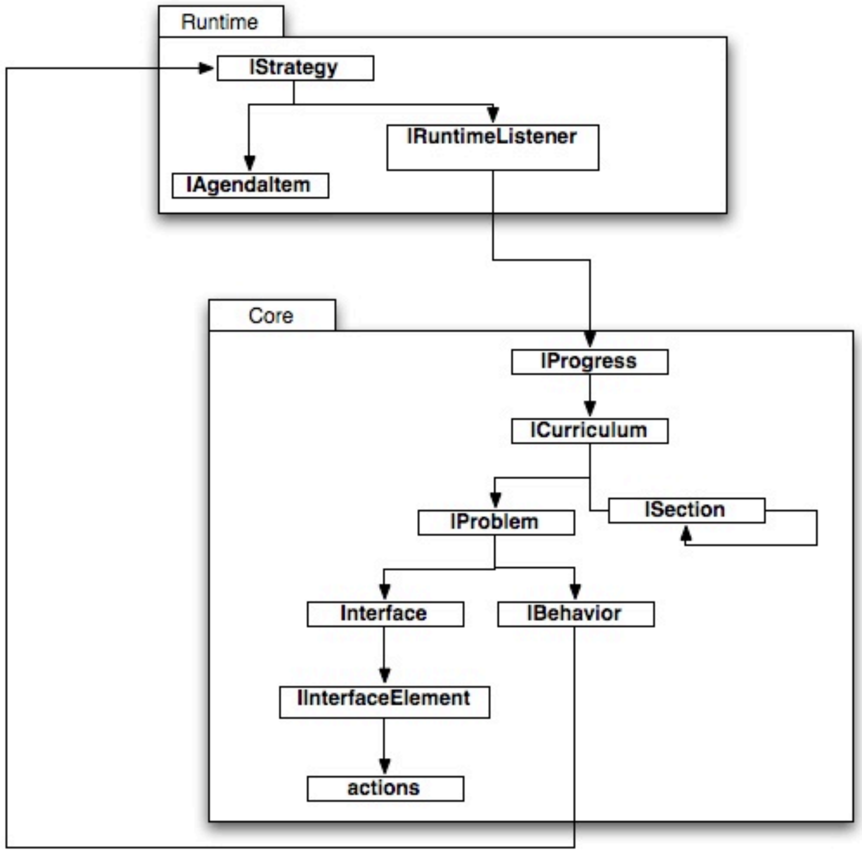


Figure 4 - Runtime Architecture

6.1.2 Use of CTOP objects in the Runtime

The runtime's first use of the CTOP objects is through the progress component, which saves a student's work in relation to a curriculum of *problems* (see Figure 4). This is the main API available from CTOP that the runtime uses to run *problems*. The progress contains indexes into the *curriculum* and its *sections* and allows a student to resume his or her work, including partially completed items. The *curriculum* and *sections* are one way that the CTOP provides extensible flow of control. Each *section* that was previously mentioned will behave differently in similar situations; such as a random *section* will provide every student with a unique ordering of *problems*. We are currently performing research on new *section* types, including a dynamic *section*, which will contain a unique set of *problems* (not just order). These *problems* will be chosen based on a set of skills that might be required to answer the *problem*, and the student's known strengths and weaknesses.

As described above, problems are composed of *behaviors* and interfaces. A problem is the second API available to the runtime. The runtime consumes and displays the output provided by the *problem's interface,* as well as translating student actions to the *problem's behavior*.

The runtime has an event model for handling incoming student *actions* (see Figure 5). Student *actions* are received as primitive XML messages that are translated into a consumable (by the various components) *action*. Each primitive *action* message is associated with an *interface* element that produced the action. The runtime uses the *agenda* in order to retrieve the associated *interface* element. This *interface* element translates the primitive action into a realized object. The runtime then passes this *action* object to the problem's *behavior*. The *behavior* object then uses this action to affect the *problem* appropriately. If it is an incorrect answer it may use *tutor strategies* to place scaffolding questions or buggy messages into the agenda. If it is correct, the runtime will just move onto the next agenda item.

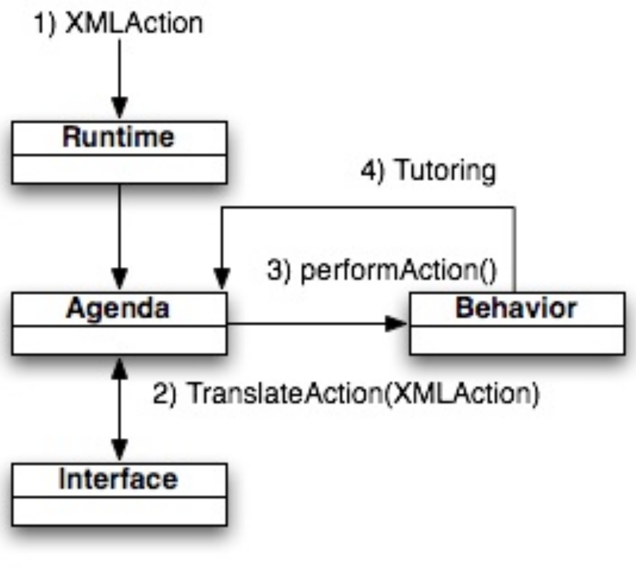
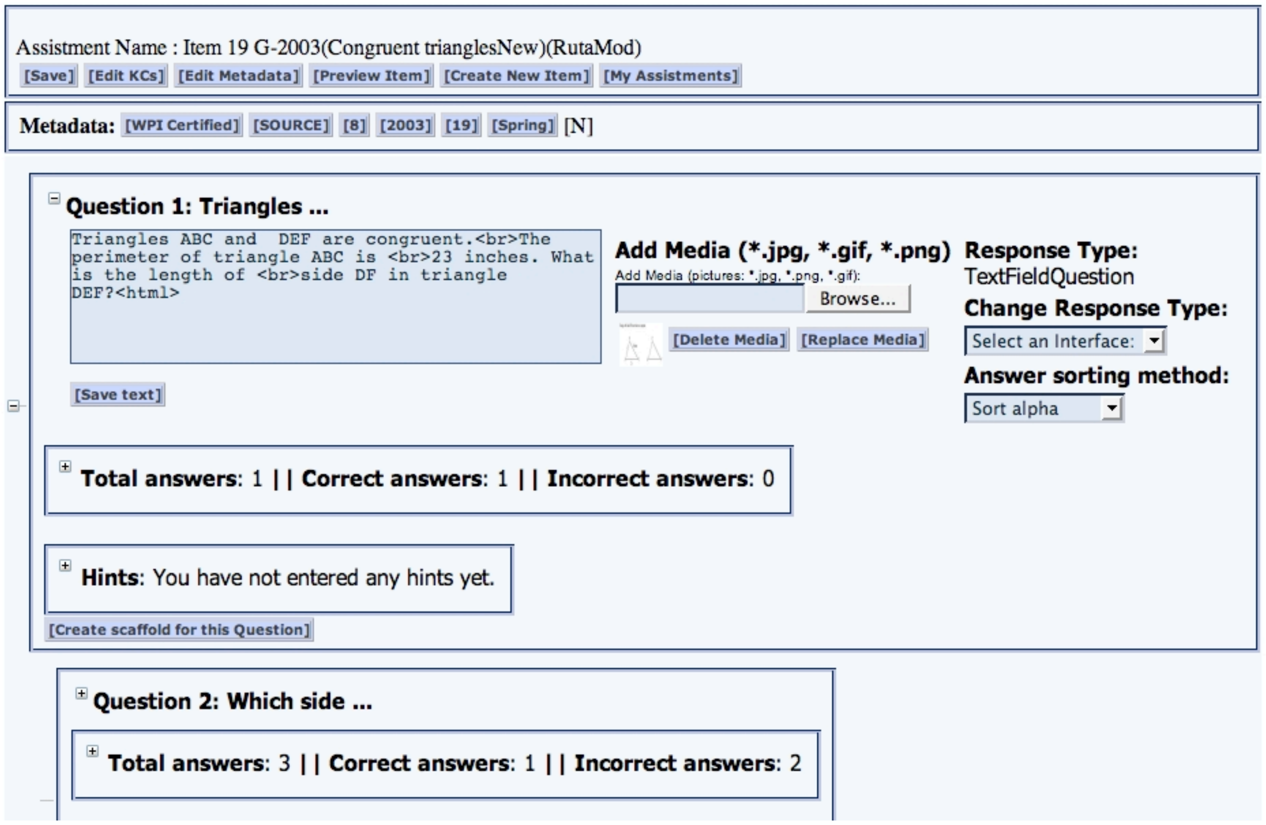


Figure 5 - Action Lifecycle

As described in earlier sections, *interfaces* contain "high-level" *interface* elements. These *interface* elements can produce a "low-level" output. This primitive output is sent to the runtime as an XML message. It is the job of the runtime to pass this XML to an *interface* display application, which produces interfaces for specific platforms. At present we have implemented a Java Swing and a HTML *interface* display application. The use of this low-level output allows the runtime to be ported to many different platforms.

6.2 Assistment Builder

The Assistment Builder (see Figure 6) was created as a web application for rapid development of content for the Assistment Project [18]. The Assistment Builder operates on the *problem* component, as well as on its sub-component *behavior*, *interface*, and properties. The Assistment Builder also provides methods for setting application-specific preferences.


Assistment Name : Item 19 G-2003(Congruent trianglesNew)(RutaMod)

[Save] [Edit KCs] [Edit Metadata] [Preview Item] [Create New Item] [My Assistments]

Metadata: [WPI Certified] [SOURCE] [8] [2003] [19] [Spring] [N]

Question 1: Triangles ...

```
Triangles ABC and  DEF are congruent.<br>The
perimeter of triangle ABC is <br>23 inches. What
is the length of <br>side DF in triangle
DEF?<html>
```

[Save text]
Add Media (*.jpg, *.gif, *.png)
Add Media (pictures: *.jpg, *.png, *.gif):
Browse...
[Delete Media]
[Replace Media]

Response Type: TextFieldQuestion

Change Response Type: Select an Interface:

Answer sorting method: Sort alpha

Total answers: 1 || **Correct answers**: 1 || **Incorrect answers**: 0

Hints: You have not entered any hints yet.

Question 2: Which side ...

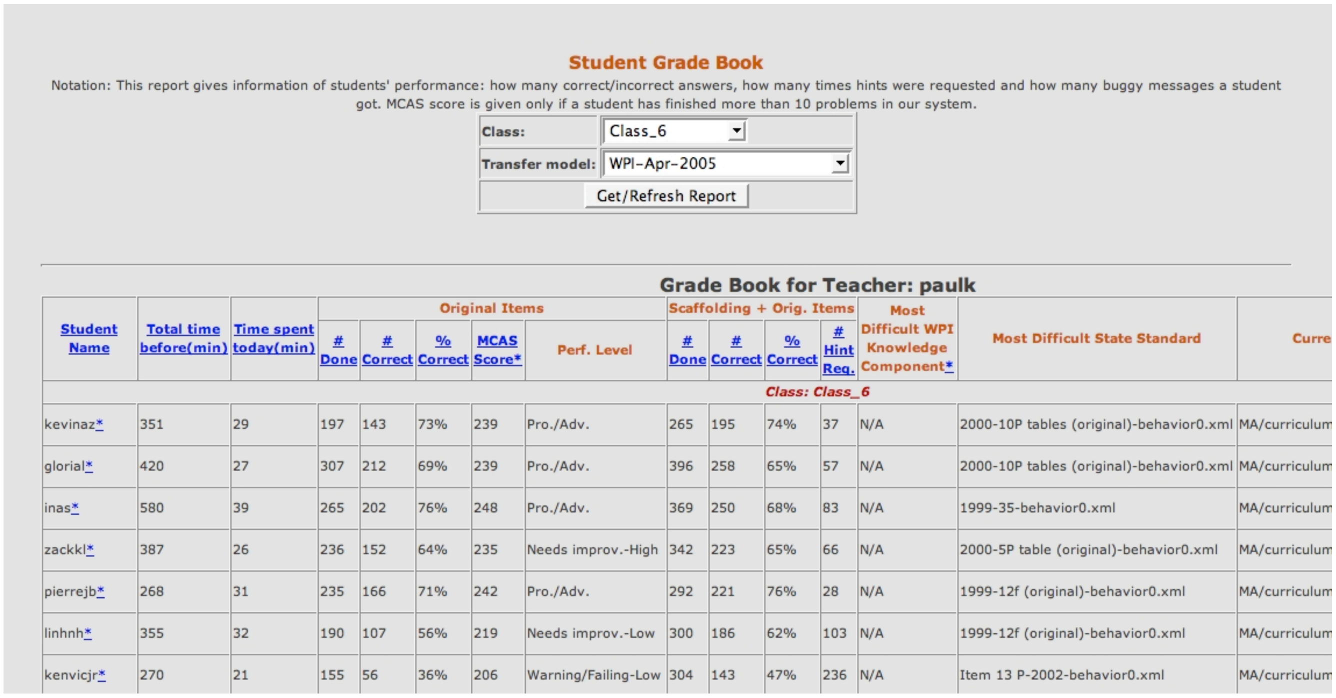
Total answers: 3 || **Correct answers**: 1 || **Incorrect answers**: 2

Figure 6 - Builder

The primary responsibility of the Assistment Builder is providing a user interface for modifying a problem's *behavior*, *interface*, and properties. The Assistment Builder does this by presenting the user with pages containing forms representing the relevant configurable parts of each of these components. As explained above a problem's *interface* is displayed for viewing and interaction with the user and is made of high level *interface* elements. The Assistment Builder uses the *interface* API to specify which high level widget is used for interacting with the user. Another manner in which the Assistment Builder uses the *interface* API is by adding the *problem's* answers as a component of the *interface*. The Assistment Builder uses the Behavior API for creating a state graph linking states and strategies using actions produced by the *interface*. The Assistment Builder allows a user to change a problem's *behavior* by specifying which *strategy* should be taken upon an answer *action*. Message *strategies* are represented as hints and "buggy messages" (messages presented if the user selects an incorrect answer) or hints, and scaffolding strategies are represented by questions nested in a tree structure. Furthermore, the Assistment Builder maintains the coupling between the *behavior* and the *interface* by modifying the *interface* whenever a *strategy* is changed in the *behavior*.

6.3 Assistment Reports

The primary goal of the reporting tool [6] is to relevantly relate each *problem* to a set of skills or concepts and then communicate that information to teachers based on their individual students. These skills, or concepts are then arranged in a hierarchy of what has been termed *knowledge components*. This hierarchy of *knowledge component* is a *transfer model*, and provides a detailed cognitive model of the *problems* being mapped to. At present, the project has completed a *transfer model* for 8th grade MCAS items and leverages this knowledge slightly in our reporting. However, the creation of larger and more detailed *transfer models* such as 10th grade math, as well as improved tools for utilizing these cognitive maps is an obvious next step.


Student Grade Book

Notation: This report gives information of students' performance: how many correct/incorrect answers, how many times hints were requested and how many buggy messages a student got. MCAS score is given only if a student has finished more than 10 problems in our system.

Class:	Class_6
Transfer model:	WPI-Apr-2005

Grade Book for Teacher: paulk

Student Name	Total time before(min)	Time spent today(min)	Original Items					Scaffolding + Orig. Items				Most Difficult WPI Knowledge Component*	Most Difficult State Standard	Curre
			# Done	# Correct	% Correct	MCAS Score*	Perf. Level	# Done	# Correct	% Correct	# Hint Req.			
Class: Class_6														
kevinaz*	351	29	197	143	73%	239	Pro./Adv.	265	195	74%	37	N/A	2000-10P tables (original)-behavior0.xml	MA/curriculum
glorial*	420	27	307	212	69%	239	Pro./Adv.	396	258	65%	57	N/A	2000-10P tables (original)-behavior0.xml	MA/curriculum
inas*	580	39	265	202	76%	248	Pro./Adv.	369	250	68%	83	N/A	1999-35-behavior0.xml	MA/curriculum
zackkl*	387	26	236	152	64%	235	Needs improv.-High	342	223	65%	66	N/A	2000-5P table (original)-behavior0.xml	MA/curriculum
pierrejb*	268	31	235	166	71%	242	Pro./Adv.	292	221	76%	28	N/A	1999-12f (original)-behavior0.xml	MA/curriculum
linhnh*	355	32	190	107	56%	219	Needs improv.-Low	300	186	62%	103	N/A	1999-12f (original)-behavior0.xml	MA/curriculum
kenvicjr*	270	21	155	56	36%	206	Warning/Failing-Low	304	143	47%	236	N/A	Item 13 P-2002-behavior0.xml	MA/curriculum

Figure 7 - Gradebook Report

The reporting application is, in fact, a multitude of smaller applications, many customized to their own specific report. However, they have a common touch point in some of the CTOP objects. *Actions* are, of course, the base component operated on by the reporting application; they are the target of most of the analysis of the myriad reports. Most of the reporting tools available rely on the *Transfer Model* components to relate *problems* to concepts. These mappings allow reports to be organized and explored by concept, as well as by teachers to evaluate the knowledge of their students in this manner. Many reporting sub-applications also use *problem*, *curriculum,* and *behavior* components to further sort, categorize or otherwise organize reporting information.

The reports themselves are all web based (see Figure 7), providing teachers and educational researchers within the Assistments Project live access to student data. The reporting applications are security and privacy conscious, allowing no confidential material to be shared outside of the classes they belong, but also allowing useful system wide reports to be shared among teachers and researchers.

6.4 Transfer Model Constructor

The Transfer Model constructor is a application presently under development by the Assistments project. It is a desktop application, relying on the *transfer model, problem,* and *interface* components of the CTOP. The constructor will be used to assemble, view and manipulate entire *transfer models* as graphs. While the Assistment Builder (see above) provides some means for the manipulation of *transfer models*, this will provide a more comprehensive tool. This application is undergoing rapid development and a prototype is anticipated before the end of 2005.

6.5 Portal

The Assistment Portal is the gateway to the Assistment Project via the World Wide Web and houses several smaller applications. The Portal focuses on the systems users and provides a means of accessing all aspects of the system. The primary objectives of the Portal are to maintain privacy and security, as well as enable collaboration among teachers. CTOP provides functionality to the Portal in the form of curriculums, problems, and a preference engine.

Every user that wishes to use our system is required to have a username/password in order to login. Once logged in, they are directed by the Portal to areas of the system they have permission to view. In addition to this level of security every application in the Portal and throughout the system verifies that the user is allowed to access this application. This is done to prevent users from simply logging into our system and then entering the URL for an application instead of utilizing the navigation provided by the Portal. System permissions are determined by the groups to which a user belongs. If a user is a member of multiple groups that conflict with each other the user's permission are derived from the group that provides them the most access.

Collaboration is also an important focus of the Assistment Portal. Users who can participate in a collaborative setting are content creators and group owners (typically teacher users). Content creators are able to collaborate by sharing created Assistments and *curriculums* with other system users; while this collaboration primarily takes place between users within a particular school it is not limited to school level collaboration. When creating shares, a user can also specify access levels to that content. In addition to explicitly created shares, there is a Released Assistment pool that is, by default, shared with every content creator. This pool is defined by the Assistment Project Team and consists of high quality items; users have read-only access to this content. If content is shared, regardless of permission level, it is then available to be utilized by any user in the share for his/her curriculums and assignments. Content that is shared as writable may be modified by any member of the share. Collaboration enables content creator to share their ideas and strategies, which in turn, allows authors to perfect their techniques and produces increasingly better and more effective content.

The smaller applications housed in the Assistment Portal are the Assistment Browser, Curriculum Manager, Assistment Finder, and Class Manager. Each of these applications provides a specific function that enables users to effectively create content and manage their classes.

6.5.1 Assistment Browser

The Assistment Browser provides a means for content creators to view, edit, and share their developed content. The browser acts on groups of *problems*, defined in CTOP, and allow users to markup their content with metadata that provides meaningful relationships among *problems* and Knowledge Components, as well as relationships between similar *problems*. From the browser it is possible to invoke the Assistment Builder application to which the problem is passed for editing. The ability to preview a *problem* is also provided to allow a user the ability to quickly review content.

6.5.2 Curriculum Manager

The Curriculum manager is an application concerned with the creation, modification, deployment, and sharing of *curriculums*. *Curriculum* objects, provided by CTOP, are created by users from any *problems* they have access to, which may include content which they authored, content that is shared, as well as officially released *problems*. In order for a *curriculum* to be used by students in the system it must be deployed and the Curriculum Manager provides an interface from which that can be accomplished. Teachers can assign a *curriculum* created by them or from a shared resource to one or more of their classes. Once a *curriculum* is assigned the students in a particular class can begin to work on that assignment. Results from the students' interactions with the *curriculum* can immediately be seen in the Assistment Reporting [6] system. Sharing of *curriculums* functions in the same manner as sharing of *problems* from the Assistment Browser.

6.5.3 Finder

The Finder is a simple search tool that is available for users to search over the vast amounts of materials for which they have access. The finder is able to locate *problems*, *curriculums*, users, and groups/classes. This tool is especially effective if a user only remembers or knows only a small amount of information about some viewable content. Permissions are strictly enforced in the Finder to ensure users are only able to search over materials to which they have access. The Finder presents results to users such that they can be loaded into the associated application.

6.5.4 Class Manager

The Class Manager is provided primarily as a means from which teachers can administer their classes. From the class manager users are able to view all their classes, view shared classes, share their classes, add classes, add students, drop students, and markup students. The idea behind shared classes is primarily for sharing of data and student results. However it also allows for users to be able to administer other users classes. This functionality allows for teacher aids and supervisors to better interact with classes under their control. Additionally, it allows schools to mimic their department hierarchies in the system, allowing for a synchronization of classes.

7. RESULTS

7.1 Framework Use

It is difficult to empirically assess the impact of the CTOP framework on development time and ease. However, there is abundant anecdotal evidence that this component framework assists in the development of new ITS applications.

The CTOP framework was developed specifically for the three applications mentioned above, the runtime, reporting, and builder. These applications had existing versions before the inception of CTOP [14][6][18], but understandably, required significant revision to operate on the new framework. The respective developers accomplished this revision in a relatively short period of time, a matter of weeks. It is also important to note that the developers accomplishing the revisions were not the original developers of most of the applications. Given the size and complexity of these applications, this is an encouraging anecdotal result on the developer usability of the framework.

In terms of CTOP maintenance and extensibility, the Datalayer provides a strong example of how the component nature allows extension. During scalability testing, the Datalayer component employed a backend relational database via Hibernate for persistence of CTOP components. As testing was scaled upward, this configuration proved unstable, and it was deemed unusable in the long term. The Datalayer was then replaced with a custom persistence scheme to improve performance. This replacement was done seamlessly, in the span of days, and required no rewrite of existing applications.

Problem definitions and *interface* element extensions are prime targets for extension within CTOP. Developers on the Assistments Project have already extended new *interface* elements, making them available to the myriad of applications. This includes a "fill-in-the-blank" multi-answer widget, as well as a ranged answer field.

7.2 Runtime Scalability

One of the goals of the Assistments Project is to provide its instructional content to many students across Massachusetts and eventually other states. To this end, the content deployment or runtime (as well as other applications) must be scalable. Since the runtime application is perhaps the application with the most existing dependencies on the CTOP, this is a prime target to test the scalability of CTOP itself.

7.2.1 Methods

To test scalability, the current production servers of the Assistments Project were used during off-peak hours. A simulation of a student logging into the Portal application, selecting a *curriculum*, and proceeding through a sequence of *problems* was recorded via JMeter [4]. This simulation was then conditioned on bounded randomized timing between student actions and requests, to more closely approximate reality. This recorded simulation was then run back, again using the JMeter software, with another bounded random start time (a few seconds). This simulation could then be scaled up via JMeter to simulate hundreds of users replicating the actions of students using the runtime.

The servers being used were both 3-gigahertz dual Xenon processors with 4 gigabytes of RAM. The application server being used was Apache Tomcat 5.0.28 with 2 gigabytes of memory allocated to its Java virtual machine. The Tomcat thread limit was pushed to 1000, and max spare threads were increased to 100. The database server was of the same hardware specification, and running a relational database optimized for transaction processing. The runtime and CTOP software was all installed on the application server machine, which is a possible bottleneck.

7.2.2 Results

The results from the JMeter simulations were encouraging. Up to 200 concurrent users simulated without an end-user performance decrease. This is indicated by an average of 2.5 second request response time. At approximately 400 concurrent users, some operations, such as problem inflation on a student proceeding to the next problem, suffered from a slightly decreased response time (averaging nearly 5 seconds). This is likely due to a bottleneck at the connection pool for inflating *problems* from the Datalayer. At 600 concurrent users, the same operation continued to be the most significant bottleneck (average at approximately 7 seconds overall, but spiking up to 30 seconds for some requests), but some other operations also had increased response time, though not to the same extent. Memory and processing consumption on the application server were not a significant concern. The database instance and its server machine were reliable and unstressed by the concurrence.

These observations imply that the only bottleneck seems to be the application server connection pool, which is easily overcome with a cluster of application servers. Even given these limitations, the current dual server setup could support a large quantity of students, perhaps as many as a quarter of the active students in Massachusetts. This estimate is achieved via the number of eligible students in Massachusetts (100,000) using the system every 10 days, students spread over 7 periods yields roughly 1500 users at any given time. To support this (given present scaling), four pairs of application server/database machines would be needed. In terms of current usage, the Assistment system presently supports over one thousand students, spread across six schools and three towns. These students are under the instruction of twenty teachers who use the reporting applications to monitor student progress and activity. These results are highly encouraging in regard to the scaling potential of the runtime and CTOP in the present and the long term.

7.3 Content Development Results

The Assistment Builder collects log data associated with content that is created by authors. This data is then analyzed and the results are used, in part, to determine the total cost of content creation and deployment in the Assistment System. While the analysis is ongoing the current results are promising, and reflect the usage of the CTOP.

Previously log data was collected on fourteen *problems* as reported in [18]. The data suggested an approximate time of 90 minutes to create *problems* ready for use. Currently there is log data for 271 completed *problems*. While these data are still being analyzed our initial findings suggest similar numbers. Of the 271 logged *problems,* not all are considered release quality. Work is currently being done to extract information from these logs about creating *problems* of release quality. This would include time spent outside of the actual builder application performing tasks such as planning and editing images, as well as organizing *problems* into *curriculums* for class assignment.

8. CONCLUSIONS

With the development of the CTOP, the Assistments Project continues to move forward, providing useful tools to teachers and students. As this project continues to be the driving force behind the CTOP, the development and scalability success of the platform are quite pleasing.

The CTOP will continue to be adopted and revised as a means to extend the Assistments Project, but could also be provided to the larger ITS community as well. The CTOP itself is a very flexible platform, and although it does not seek to provide all the services a full component framework does, it can be quite powerful.

8.1 Future Work

As mentioned previously, there are other applications and extensions presently being developed with the Assistments Project. These include extensions to support Bayesian inference for problem selection within the runtime, additional reports, as well as an integrated curriculum development and reporting tool. Additional collaborative tools are also forthcoming, allowing content authors who use the Assistment Builder to easily manage and deploy their work, while collaborating with other authors.

Yet another future possibility is the ability to offload the evaluation of a *problem*. This will enable the Assistment System to send the user's answer to a remote server for evaluation, taking the load off of the main web servers. In addition, we will be able to support the evaluation of questions that the Assistment System is not capable of evaluating. One can imagine a scenario, under which an author has a working Java code verification system that can be used to evaluate the student's response to a particular Java question. The author will be able to specify which remote server to send the students response to, the remote server will evaluate the code entered by the user, and a response will be sent back to the Assistment System. The response is then displayed to the user or directs the Assistment System to the next course of action.

9. REFERENCES

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

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

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

[4] Apache JMeter. (2005). http://jakarta.apache.org/jmeter/

[5] Feng, M., Heffernan, N.T, Koedinger, K.R. Addressing the Testing Challenge with a Web-Based E-Assessment System that Tutors as it Assesses . Submitted to WWW2006, Edinburgh, Scotland (2005)

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

[7] Hibernate Relational Mapping Tool. (2005). http://www.hibernate.org/

[8] Jackson, G.T., Person, N.K., and Graesser, A.C. (2004) Adaptive Tutorial Dialogue in AutoTutor. Proceedings of the workshop on Dialog-based Intelligent Tutoring Systems at the 7th International conference on Intelligent Tutoring Systems. Universidade Federal de Alagoas, Brazil, 9-13.

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

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

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

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

[13] Murray, T. (1999). Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art. International Journal of Artificial Intelligence in Education, 8, 30-43.

[14] Nuzzo-Jones, G., Walonoski, J.A., Heffernan, N.T., Livak, T. (2005). The eXtensible Tutor Architecture: A New Foundation for ITS. In Proceedings of the 12th Annual Conference on Artificial Intelligence in Education 2005 Workshop on Adaptive Systems for Web-Based Education: Tools and Reusability, Amsterdam

[15] Published by Carnegie Mellon University (2003), Open Learning Initiative (OLI), http://www.cmu.edu/oli/

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

[17] Rose, C. P. Gaydos, , A., Hall, B. S., Roque, A., K. VanLehn, (2003), Overcoming the Knowledge Engineering Bottleneck for Understanding Student Language Input , Proceedings of AI in Education.

[18] Turner, T.E., Macasek, M.A., Nuzzo-Jones, G., Heffernan, N..T, Koedinger, K. (2005). The Assistment Builder: A Rapid Development Tool for ITS. Poster in the 12th Annual Conference on Artificial Intelligence in Education 2005 Workshop on Adaptive Systems for Web-Based Education: Tools and Reusability, Amsterdam

[19] Vendlinski, T., Niemi, D., Wang, J., Monempour, S., Lee, J. (2005).Improving Formative Assessment Practice with Educational InformationTechnology. American Educational Rsearch Association 2005 Annual Meeting.

[20] Walonski, J.A. & Heffernan, N. T. (2005) Towards Improving the Assistment System by Tracking Student Off-Task Behavior. In preparation for publication.