

Instructor Authoring Tool:

A Step Towards Promoting Dynamic Lecture-Style Classrooms

by

Jessie I. Chen

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

February 9, 2006

Copyright 2006 Jessie I. Chen. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author _____

Department of Electrical Engineering and Computer Science
February 9, 2006

Certified by _____

kimberlie Koile, David Cavallo
Thesis Supervisors

Accepted by _____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

BARKER

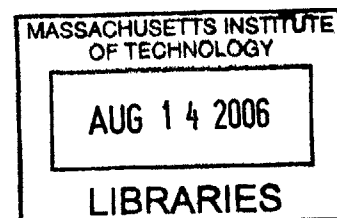


TABLE of CONTENTS

Abstract	3
Section 1: Introduction/Background	4
Section 2: This Research – A Better Instructor Authoring Tool	11
Section 3: Approach	12
Section 4: The Instructor Authoring Tool (IAT) – System Overview	19
4.1 User’s View of System: Sample Scenario	20
4.2 Implementation	22
4.2.1 Creating an Add-in to PowerPoint	23
4.2.2 Graphical User Interface	23
I. Controls	24
II. Inserting an Exercise Slide	24
III. Committing an Exercise	36
IV. Displaying List of Exercises, Exercise Details, and Editing Exercise Answers	37
V. Saving Exercises to the Database	46
4.2.3 CLP-Wide Classes	46
4.2.4 IAT-Wide Classes	51
4.2.5 Integration with “InstructorModeAddin”	54
Section 5: Future Work	56
Section 6: Results and Conclusion	58
References	59

Instructor Authoring Tool:
A Step Towards Promoting Dynamic Lecture-Style Classrooms
by
Jessie I. Chen

Submitted to the
Department of Electrical Engineering and Computer Science

February 9, 2006

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The Instructor Authoring Tool of the Classroom Learning Partner project provides an efficient tool for university professors and other teachers of lecture-style classrooms to construct lecture slides that can easily accommodate effective in-class exercises. In designing such an authoring tool, five criteria were taken into consideration: An enhanced ability to support instructors' use of in-class exercises by facilitating the slide-generation process, the use of Microsoft PowerPoint as a basic tool, enabling real-time feedback of student performance on exercises, the leveraging of past student mistakes and misunderstood concepts in enhancing teaching, and the support of an automatic tutoring system to be implemented at a later time. These criteria were successfully met by the resulting instructor authoring tool, though improvements towards facilitating the slide-generation process and support for real-time feedback of student performance have yet to be tested in a formal academic setting.

Thesis Supervisors: Kimberle Koile, David Cavallo
Titles: Research Scientist, MIT Computer Science and Artificial Intelligence Lab;
Research Scientist, MIT Media Lab

Section 1: Introduction/Background

Worldwide Challenge: Better Use of IT Technology for Addressing Educational Needs

Countries all around the world are making increasingly large monetary investments into the field of educational technology. For 2002-2003, the United States alone estimated 12.1 billion dollars in technology spending for K-14 education, and worldwide corporate external training spending is forecasted to grow to more than \$80B by 2006 (Hinrichs 2002). However, as Senator Maria Cantwell writes in her Amendment to America's Bill on Education concerning a February 2001 Net Day study, "...97 percent of teachers have access to some computer technology, but only 32 percent were integrating computers in classroom learning" (Hinrichs, 2002). This discrepancy starkly highlights our collective need, in the words of Randy Hinrichs of Microsoft's Learning Science and Technology Group, "to start creating learner centric technologies.....[that] shift... away from only lecture based environments with pen and paper assessment...and... [somehow] use technology to extend the classroom, making instruction more flexible for the learner,...mentor, and ...instructor while focusing on the experience of learning" (Hinrichs 2002). According to Hinrichs, "Introducing technology into the classroom is one thing, introducing it effectively is another" and "we concentrate too much on recreating [rather than extending] the classroom" (Hinrichs 2002). Similarly, Professor William Graves of Bryant University in his *Framework for an e-Learning Strategy* also states that, "Indeed, the needs of a learning society will not be met until virtual technologies are used, not just to make instruction more convenient and accessible, but also more effective...This is the most

immediate and pressing challenge for most campuses...[of] Higher Education." (Hinrichs, 2002). In a recent US publication of the President's IT Advisory Committee on Using Information Technology to Transform the Way We Learn, the collective authors consistently suggested that "industrial experience over the past two decades demonstrate that successful IT-assisted process improvement almost always requires that IT be coupled with careful rethinking of the targeted processes and social institutions" (President's IT Advisory Committee). We as innovators, therefore, have an immediate and pressing responsibility to search for and arrive at effective novel applications of IT technology that can revamp the roadmap of 21st century education to an extent that is in line with the potentials and needs of our social institutions .

Classroom Presenter (CP): Addressing the Challenge

One attempt to address this challenge is the creation of Classroom Presenter (CP) by the ConferenceXP research group of Microsoft Corporation. The problem that motivated the work of CP was how to improve the ability of an instructor to present lecture material from a computer. Although there are significant advantages to computer projection of lectures such as the ability to prepare high quality examples in advance, the ease of switching between slides and web content or other applications, and the ability to share and reuse material (Bligh 2000), these advantages often come at the expense of flexibility during presentation, causing lectures to become highly scripted. The goal of CP is to address these inconsistencies in a presentation system suitable for both large lectures and distributed classes." (Anderson et al 2004, 2005). After much research and experimentation, the Classroom Presenter arrived at some significant conclusions,

including 1) The importance of integrating speech, ink, and slides (Anderson & Hoyer et. al.), and 2) the importance of separating the control view from the display view by using separate machines (Anderson et. al. 2004).

The Importance of Feedback and Unsuccessful Implementations

A third crucial finding of the Classroom Presenter project is the confirmation of the importance of student feedback. According to Prof. Richard Anderson of the University of Washington and founder of ConferenceXP, "Student-instructor interaction is vital to student learning, but soliciting student feedback in large, university-level lecture classes is challenging," and "As universities serve more students and face tighter resource constraints, these large lectures are likely to persist, necessitating innovative approaches to large class challenges (Anderson et. al. 2005) ." Hinrichs also states that, "The underbelly of integrating technology into the classroom is the ability to capture the learner's experience and to provide adequate feedback and response" (Hinricks 2002). This includes facilitating both a student experience that enhances the student's ability to communicate and collaborate, and a teacher experience supported by active presentation learning services (Hinrichs 2002). For example, students should be able to write directly on PowerPoint type applications during lectures, or during review of materials. Teachers should be able to receive this feed back either real time or post lecture and make adjustments to their instruction, to their materials, or otherwise influence their student's learning behavior (Hinrichs 2002). Ideally, individual assignments can then build on teacher's observing how students understand facts, concepts, procedures, processes and principles as they're introduced. Over time, systems have indeed been built to enable

student feedback including the ActiveClass Project, which provided functionality for students to submit questions (Ratto, et al. 2003), and eFuzion, which allowed students to post questions and answers to a group website (Peiper). Nonetheless, ActiveClass-style systems were generally unsuccessful in large classes because students tended to ask either few or unrelated questions. In the summer of 2002 when eFuzion was deployed in a classroom at the University of Illinois, it was reported to have improved the final grades of students by approximately 6 points (Peiper). However, despite such apparent success, systems such as eFuzion have proved to be distracting to both instructor and students as they supported multiple simultaneous conversations (Davis 2005, Grimson 2005).

Classroom Feedback System (CFS): Promising Implementation

The Classroom Feedback System (CFS) is one promising implementation that emerged for addressing the need for student feedback. Researchers for CFS identified from literature and from experimentation with prototypes of CFS that there were four primary factors inhibiting student-initiated interaction in large classrooms (Anderson 2003, Brown 1992). These were 1) Feedback Lag: Students doubted the value of their questions on a topic until the topic was closed, but felt the chance to ask their questions had passed once the topic moved on, 2) Student Apprehension: Fear of speaking due to the size or climate of the class, 3) Single-Speaker Paradigm: Model in which only one person (student or instructor) speaks at a time which does not scale to broad participation in large classes, and 3) Comment Verbalization: Students have trouble communicating their confusions in words (Anderson et. al. 2003, 2005). Based on these findings, the CFS system was implemented to allow student generation of feedback from a fixed list of possible annotations (ie. MORE

EXPLANATION, GOT IT, EXAMPLE, etc) by right-clicking and then selecting a category from a menu, which is then sent to the instructor's device and removed by the student once the issue has been addressed. the implementation and testing of CFS resulting in the discovery of several advantages to CFS-style student feedback systems: 1) CFS was successful in promoting classroom interaction and directly addressed each of the four primary factors inhibiting student-initiated interaction (Anderson, et al. 2003), 2) CFS created the possibility for student-guided lectures (Anderson, et al. 2003, "Interaction"), and 3) feedback from CFS created the possibility for instructors to postpone feedback and loggable data promotes flexibility for addressing student needs (Anderson, et al. 2003, "Interaction").

Classroom Assessment Techniques (CATs): Instructor Centered to Student Centered

While CFS was able to make progress in improving educational interaction in large lecture halls, it was still highly instructor-centered; The feedback largely contributed to facilitating the instructor's delivery of material, but did not take into account prior knowledge and experiences students bring into the classroom. To address this shortcoming, Sarah Schwarm and Tammy VanDeGrift from the University of Washington attempted to use Classroom Assessment Techniques (CATs) to elicit the process students used to construct of knowledge, so that that very process can be shaped and guided to facilitate learning (Schwarm 2002) . While CATs proved useful in helping educators understand how students are making connections between concepts and existing knowledge, and thereby helped move the university lecture hall another step towards being learner-centered, there is still one fundamental change that needs to occur at the heart of the large lecture: The

integration of real-world problem solving.

Solving Real-World Problems and Beyond

David Merrill, one of America's leading pedagogues, indicated the following elements as necessary for making any educational system effective: 1) New knowledge is demonstrated to the learner, 2) Existing knowledge is activated as a foundation for new knowledge, 3) New knowledge is applied by the learner, 4) Learners are engaged in solving real-world problems, and 5) New knowledge is integrated into the learner's world (Hinrichs 2002). While number one can easily be achieved through the lecture format, and number two through incorporation of concepts like the CATs, we have yet to find a way of facilitating three, four, and eventually five in the most common educational setting in which students find themselves: the large lecture hall.

Seymour Papert of the MIT Media Lab's Future of Learning Group once made the following comment:

Being a mathematician is no more definable as "knowing" a set of mathematical facts than being a poet is definable as knowing a set of linguistic facts. Some modern math ed reformers will give this statement a too easy assent with the comment: "Yes, they must understand, not merely know." But this misses the capital point that being a mathematician, again like being a poet, or a composer or an engineer, means *doing*, rather than knowing or understanding. (Papert, 1971)

In the same way, the large lecture hall has historically, to date, been prone to emphasizing knowledge, and even understanding, at the expense of application. This method of teaching highly curtails the healthy development of a student on every level, whether intellectual, emotional, spiritual, or otherwise, and fails to integrate new knowledge into

the vastly multidisciplinary world in which we live. Even on a most practical level, Hinrichs points out that

Neither university, nor industry has utilized the technology yet to enable our engineering workforce to enhance their skills for employability. According to the National Standards Skills Board, those skills include "listening, speaking, using information technology and communications, gathering and analyzing information, analyzing and solving problems, making decisions and judgments, organizing and planning, using social skills and adaptability, working in teams, leading others, building consensus, and self and career development". The[se] employability skills are universal.....[but] "We're focusing on the technology, not the learning" (Hinrichs, 2002)

Hindrichs goes on to say that, "Lecturing is important and economical for conceptual transfer, but learning by doing requires students to improve their performance to achieve certain tasks. Project based learning that involves problem solving is learner centric and can be enabled with today's communication and collaboration software" (Hindrichs 2002). Overall, much of the current work in cognitive psychology has shown that students learn better when engaged in solving problems, using problems that are be authentic, real world, and, if possible, personal (Hindrichs 2002). (Problem based learning of the type Hindrichs recommend is well represented by a number of recent instructional models including: Collins et al (1989) Cognitive Apprenticeship; Schank et al (1999) Goal Based Scenarios; Jonassen (1999), Constructivist Learning Environments; Savery & Duffey (1995) Problem Based Learning; Clark & Blake (1997) Novel Problem Solving; and van Merrienboer (1997) Whole Task Practice in 4C/ID Model (Hinrichs 2002).)

Personal Response System (PRS)

One method by which instructors can begin to incorporate the application of material learned in lecture (Merrill's third principle), is through the use of in-class exercises. Using

students' answers to in-class exercises, specific misunderstandings can often be identified through direct mapping to incorrect answers. The Personal Response System (PRS) is one example of a system using this technique of formative assessment that has proven successful in both small and large classroom settings. In PRS, students use a transmitter to submit answers to multiple-choice, true and false, or matching questions. The results are then tabulated and displayed on the instructor's computer in the form of a histogram (Draper 2004). While providing a way for the application of new material through in-class exercises, the PRS limits exercise types to multiple-choice, true and false, and matching questions. Currently, a greater variety of in-class exercises may be an option only in classrooms with a small enough class size so that instructors will be able to manually assess student work performed on blackboards, paper, or tablet-pc-based systems (Simon, et al. 2004).

Section 2: This Research - A Better Instructor Authoring Tool

In light of the current evolution of applied educational-IT, this MEng thesis project proposes the creation of an enhanced Instructor Authoring Tool (IAT) for incorporation into a larger project called the Classroom Learning Partner (CLP) which aims to extend Classroom Presenter support for expanding possibilities in teaching. It will retain all the advantages of CP, CFS, CATs, and PRS, but will in addition, 1) Allow students to submit non-multiple-choice answers back to the instructor in real time, thereby enabling a greater variety of in-class exercises to be used, and 2) Aggregate the responses for an instructor-view-friendly display for classes with large numbers of students (100+), thereby enabling these in-class exercises to be used even in large lecture halls.

The goal of the IAT and CLP at large is to empower large university-style lectures by enabling more meaningful instructor-student interactions through diverse in-class exercises. The CLP will facilitate what corresponds roughly to David Merrill's third element of an effective educational system, that "New knowledge is applied by the learner." The IAT enables instructors to practice teaching that is more dynamic by providing solid groundwork for formative assessment, and encourages students to be hands-on active participants in their own learning.

Currently, the resulting IAT is able to successfully build a multi-view PowerPoint slide deck containing in-class exercises with expected answer types of "Number," "Text," "Sequence," "Set," "SchemeExpression," or "None." It is also able to export the .PPT slide deck into a .CSD file that can be compatibly used within Classroom Presenter. In addition, the IAT contains functionality for saving slide exercises to a database repository for storage and integration with other CLP modules. It is hopeful that the CLP will quickly and effectively lay the ground work necessary for incorporating Merrill's fourth ("Learners are engaged in solving real-world problems") and fifth ("New knowledge is integrated into the learner's world") elements into the large-lecture-style classes of the 21st century.

Section 3: Approach

In constructing the framework of a new IAT, the following elements were taken into account: An enhanced ability to support instructors' use of in-class exercises by facilitating the slide-generation process, the use of Microsoft PowerPoint as a basic tool,

enabling real-time feedback of student performance on exercises, the leveraging of past student mistakes and misunderstood concepts in enhancing teaching, and the support of an automatic tutoring system to be implemented at a later time.

Facilitating the Instructor Authoring Process

According to Professor of Education Larry Cuban of Stanford University, most postsecondary educators with research responsibilities lack the time to become proficient in emerging technologies or to envision the potential of technology as a teaching tool (Cuban, 2001). Although many faculty members have embraced information technology as a means of extending traditional lecture-and-text-based education systems, they often use this technology only to post syllabi and lecture notes on course Web sites, or to provide threaded discussions and chat rooms on course material (Cuban, 2001). Cuban states that, while these applications are useful, they do not “tap into the real potential for using computers to revolutionize teaching and learning” (Cuban, 2001). Similarly, recent National Research Council reports had repeatedly called for the creation of effective models, developed with the full understanding of the principles of learning, to promote interactions between technology, user-driven research, and classroom practice (Brewer, 2004). In light of this, a user-friendly authoring tool for the instructor is of prime importance.

PowerPoint as a Basic Tool

The Regents of the University of Minnesota have recently pointed to PowerPoint as an important tool for facilitating “classroom assessment,” a practice that provides instructors

feedback on what and how much their students are learning so that they may use the information gathered to measure the effectiveness of their teaching practices, make decisions, and implement changes that result in better student learning (Regents, 2006). In her research on the use of PowerPoint slides in a classroom, Belinda Ho, Associate Professor of English and Communication at the City University of Hong Kong, had also listed a multitude of advantages in advocacy of PowerPoint as an instructional tool including that “the slides cannot get lost,” “they can be easily refined and reused in the following years,” “they are flexible,” and “the presenter can print handouts with two, three, or six slides on a page,” overall bringing PowerPoint to light as an ideal starting point for a new authoring tool (Ho, 2001). Nonetheless, Ho had also listed in her paper disadvantages of PowerPoint including that “they foster more passive learning” and “they are not well-suited to drawing impromptu sketches.” These are the kind of setbacks that will also need to be address in a new authoring system based on PowerPoint.

Enabling Real-Time Feedback

In a study sponsored by the U.S. Department of Education’s Office of Educational Research and Improvement, entitled “Using Technology to Support Education Reform,” it was reported that

A well-known problem in many conventional classrooms is the mismatch between the level of presentation and the understanding of many students. As teachers describe concepts and procedures, they depend on student feedback to indicate any comprehension problems. Unfortunately, the students who understand the material best are most likely to contribute to class discussion. Students who don’t understand simply remain silent, and the instructor continues with an explanation that some students find incomprehensible” (Means, 1993).

Providing feedback to students of their current level of understanding of concepts can be critical for effective learning, as well as useful for the professor in accomplishing such goals as presenting and effectively covering a probing question at the heart of the subject matter, gathering student responses rapidly and anonymously, and quickly assembling a public, aggregate display (such as a histogram) that makes salient the variation in the group's ideas without disclosing individual contributions (Kadlowec, 2005; Roschelle, 2004). Professor Maria Satratzemi of Applied Informatics at the University of Macedonia had said the following about an educational programming environment designed there:

We believe that knowledge of the path followed by students to the solution of a problem is, usually, extremely valuable information to someone wishing to explore student conceptions about programming and problem solving techniques. The capability to systematically record such paths can open up interesting new possibilities for exploring the conceptions of students. Our educational programming environment systematically records the actions of students, thus offering teachers with invaluable information about the path to the solution followed by the students, the steps backwards, the repeated tries, the mistakes, and the hesitations. We designed an educational programming environment that records and stores what is didactically essential. (Satratzemi, 2001)

An improved instructor authoring tool will aim to support a system that will make as much of this kind of detail as possible available to instructors in a digestible format in real-time during class so that he or she may be better informed and prepared to proceed in teaching.

Leveraging Past Student Mistakes and Misunderstood Concepts

The importance of utilizing past student mistakes and misunderstood concepts in conjunction with technology in teaching has been demonstrated in a number of different ways to date. Utah State University Assistant Professor Bryan Warnick once said of projects by AI Researchers Roger Schank and Adam Neaman of Yale University, that "The

most impressive thing about [these] applications...is the care that the developers have taken to make user mistakes meaningful. Specifically, they try to simulate those conditions under which novices tend to make errors and then offer the appropriate just-in-time expert advice” (Warnick, 2003). Schank and Neaman themselves have also produced thoughtful discussions on how failures can be used to make errors both educative and motivational, and also how computer simulations can help students to make educational mistakes in a non-threatening atmosphere” (Schank, 2001). In addition, paying attention to student errors can have other benefits such as bringing about a better understanding of why some misconceptions about the world are harder to change than others (Feltovich, 2001). At the same time, a record of past mistakes can be invaluable in the creation of wrong answers for multiple-choice questions, a highly tested and endorsed method for evaluation (Higgins, 2006). Thus, it would also be important for an improved instructor authoring tool to support an overall system that makes use of past mistakes and misunderstood concepts in achieving its educational goals.

Maintaining a Record of Instructor and Student Input / Preparative Support for an Intelligent Tutoring System

Finally, there are several reasons why it would be of benefit to maintain a systematic record of inputs from both the professor and the student. First, cultural considerations will often need to be taken into account when technologies or programs enter the classroom, each with its own set of cultural entailments, representing the goals, expectations, histories, values, and practices associated with a particular community, or entailments of a community (Bouillion, 2001). When material is passed from one generation of instructors to another, or even moved geographically, it is critical that provision is made for notes and

explanations by past instructors to be attached where cultural-specific choices had been made. A record of student work will also provide a valuable means by which the development of student learning with respect to particular concepts and skills can be assessed over time, as opposed to for instance, making a single measurement at some final or supposedly significant time point (Wilson, 2004). Nonetheless, there has been very little technology developed thus far that provides a comprehensive system for inputting, storing, retrieving, analyzing, and representing performance data (Means, 1993). Such a need is worthy of address as one such tool developed in the past at the Education Development Center by Researcher Midian Kurland, *TextBrowser*, which provides an electronic analogue to teachers traditional methods for keeping track of assignments, marking student papers, providing feedback, and recording and monitoring student performance, had proven to be able to substantially enhance a teacher's ability to respond to, store, retrieve, assess, and manipulate student work (Mean, 1993).

Finally, a crucial reason for maintaining a database of instructor and student input is as preparation for an intelligent tutoring system (ITS) to be implemented in the future. Among the benefits of intelligent tutoring systems, is the support it provides for human tutors which will in turn enable them to provide more individualized help to his or her students. For example, Professors Kenneth Koedinger and John Anderson of Carnegie Mellon University reported, among the benefits derived from the use of their geometry ITS, the following: "While students were engaged in interacting with the tutor, the human tutor was free to roam around the classroom giving extra help to poorer students who needed it or challenging better students to do more than they might otherwise" (Virvou, 2000).

Furthermore, one of the major problems of a mathematics tutor in a class is that he cannot check the answers of all the students of his class simultaneously. Therefore, such a human tutor can be significantly assisted by an ITS that performs individualized error diagnosis to students' solutions" (Virvou, 2000). Tutoring systems can also be highly beneficial in enabling students to learn about selected material at their own rate before it is taught or as a supplement to course activities (Woolf, 2001). Associate Research Professor Beverly Woolf at the University of Massachusetts predicts that such systems might become routine supporting group collaborations of students-at-a-distance, exploration of hypothetical worlds, and the making and testing of hypotheses...[As a result,] learning need not be constrained to place and time" (Woolf, 2001) .

One example of a highly successful tutoring system is the *Pump Algebra Tutor* (PAT), originally developed by the Pittsburgh Advanced Cognitive Tutor Center at Carnegie Mellon University with support from NSF, Darpa, and foundations in Pittsburgh.

Professor Koedinger comments that

With PAT, students can create tabular, graphical, and symbolic models of problems. The cognitive tutor chimes in as needed with just-in-time feedback. The tutor highlights possible errors with outline text comparing student actions with its database of common errors. The student can also request a "context sensitive" hint. The tutor tracks the learner's progress on a problem and then, in a fairly complex process, selects the appropriate advice: The tutor chooses the hint message by using the production system to identify the set of possible next strategic decisions and ultimate external actions. It chooses among these based on the student's current focus of activity, what tool and interface objects the student has selected, the overall status of the student's solution, and internal knowledge of relative utility of alternative strategies. Successive levels of assistance are provided in order to maximize the students' opportunities to construct or generate knowledge of their own (Koedinger, 2003).

PAT also supports learning through "knowledge tracing," which tracks a student's

problem solving skills, identifies areas of difficulty, and presents problems in areas that have not yet been mastered. PAT seems to work well: field studies show a 15-25 percent difference between those classes that used PAT versus control groups (Koedinger, 2003). Finally, Professor Koedinger discussed how PAT has also proven a successful tool for teacher change (Koedinger, 2003). PAT offers a student-centered model of learning, that teachers, once they are exposed to it, often tend to replicate in other areas of instruction (Koedinger, 2003). Koedinger observes that, sometimes, teachers begin to borrow the problems, the representational tools, and the feedback strategies embedded in the cognitive tutors (Koedinger, 2003). He writes, “Teacher began to use PAT problems in their regular classes and began to experiment with more student-centered learning by doing outside of the computer lab...Thus, the computer becomes a pedagogical role model (Koedinger, 2003). These examples together highlight the potentials of an intelligent tutoring system if one were to be derived from a record of ongoing instructor and student input. This is ample encouragement for a new instructor authoring tool to set support in place for some form of intelligent tutoring system to be designed in detail and implemented at a future time.

Section 4: The Instructor Authoring Tool (IAT) – System Overview

The IAT System consists of an Add-in to Microsoft PowerPoint that will enable instructors to embed exercise objects within their PowerPoint slides. It consists of four main components: A graphical user interface for exercise entry, display, and modification; the tools for storing exercise objects and all associated information to a standing database used to archive instructor and student input and for integration with other CLP modules; the creation of precise and suitable classes for relevant objects to be stored to database and

shared with other CLP modules; and hooks for integration with the InstructorModeAddin created by Professor Stephen Wolfman while at the University of Washington (a PowerPoint Add-in which enables the viewing of the multiple modes of Classroom Presenter and exporting PowerPoint slides to .CSD format, the format understandable to Classroom Presenter (Simon, 2003)) so that the necessary metadata from instructor-created exercises can be passed on to the exported .CSD files for use by other modules of the CLP.

4.1 User's View of System: Sample Scenario

The following is a sample user scenario that illustrates how the IAT will be used to create a lecture slide containing an in-class exercise for 6.001 Structures and Interpretation of Computer Programs, the introductory course in Computer Science given at MIT:

- a.** Prof. Grimson clicks on the “Insert Exercise Slide” button in the toolbar and a new slide is inserted with a message prompting him to enter question text in a suggested location. Icons indicating “Keyboard,” or “Pen” also appear, and a message prompts him to choose the mode of interaction he desires for his answer box.
- b.** Prof. Grimson clicks on “Keyboard” for interaction mode and a ComboBox appears allowing him to select the expected answer type if he wishes. A rounded-rectangular answer box also appears containing the text “Text Answer Box.”
- c.** Prof. Grimson enters the question “The following expressions evaluate to what type? ($3.14 (* 2 5)$)” in the suggested question TextBox.
- d.** Prof. Grimson selects “Text” for expected answer type.
- e.** Prof. Grimson enters “#” in the answer box.
- f.** Prof. Grimson clicks on the “Commit Exercise” button in the toolbar, and the expected

type combobox as well as the original answer box disappears. Instead, the slide is left with a single box indicating the student answer area for this exercise. A tag also appears on top of the answer box marked with “TEXT...1” indicating that this is the first answer box in the slidedeck expecting an answer of type “Text.” When Prof. Grimson mouses over this tag, a mini window appears displaying the answer text “#” for this answer, along with the date on which this exercise was created, and the full text for the expected type of this answer box. In addition, a label with the text “Exercise #1” also appears on the top left corner of his slide to help him keep track of his embedded exercises.

g. Prof. Grimson predicts that many of the students might end up thinking that the scheme evaluator will return true or false when this expression is evaluated rather than an actual number, and decides it would be helpful to enter “Boolean” as a common incorrect answer.

h. Prof. Grimson clicks on the “Display List of Exercises” button in the toolbar, and a window appears listing all the exercises currently in his slide deck.

i. Prof. Grimson clicks on “Exercise#1~Slide#2~The following...” and a new window appears labeled “Exercise Details” and “Exercise #1.”

j. Prof. Grimson clicks on the “Add New Answer” button and a new blank answer row labeled “2” appears with room for him to enter answer text, answer type, whether it is a correct answer, any note or study material he would like to indicate, and a checkbox that enables him to delete the answer in the future if he ever chooses to do so.

k. Prof. Grimson enters “Boolean” as the answer text, selects “Text” from the answer type drop down menu, chooses “No” for whether the answer is correct, and under the notes section jots the note that “Student should review textbook Chapter 1, Sec. 3.”

l. Prof. Grimson then clicks on the “Apply Changes” button and the Exercise Details

window closes.

m. Prof. Grimson then creates several more exercise slides in a similar fashion and decides that he now has all the slides he need for tomorrow's lecture.

n. Prof. Grimson clicks on the "Save Exercises to Database" button, and a messagebox appears indicating to him that his exercises have been successfully saved.

o. Prof Grimson the goes to the "File" menu and selects "Export to CSD..." and a file browser appears prompting him to select the location to which his .CSD file should be saved.

p. Prof. Grimson names his lecture "Evaluation.csd," chooses to save to Desktop, and clicks on the "Save" button.

q. Prof. Grimson then decides to briefly rehearse his lecture for the next day.

r. Prof. Grimson opens up Classroom Presenter, and opens Evaluation.csd.

s. Prof. Grimson chooses the Instructor view in Classroom Presenter.

t. Prof. Grimson is happy that the labels for exercises as well as the first answer text he entered for each exercise is visible to him as he gives the lecture.

u. Just to be safe, Prof. Grimson switches to Student view to make sure that the answers to his exercises do not appear in their answer boxes (but only in his own) and is satisfied.

4.2 Implementation

The IAT is implemented as a Microsoft PowerPoint Add-in. The Add-in provides a graphical user interface for authoring exercises with any of five different expected answer types, including "None" and one for runnable scheme expressions, and also saving exercise objects to the database. The GUI consists of a standard PowerPoint toolbar, as

well as buttons with identical functionalities added to appropriate menus for user flexibility; User input is generally received through windows forms, and the IAT is merged with the InstructorModeAddin for exporting to .CSD by creating a single common connect.cs class as the basis for the resulting Add-in.

4.2.1 Creating an Add-in to PowerPoint

Formerly widely performed in Visual Basic, an Add-in to any of Microsoft Office Suite application can now be created using C# .NET within the Microsoft Visual Studios environment. To do this, the developer should look for “Extensibility Projects” under the “Other Projects” folder when creating a new project, and choose “Shared-Add-in.” After naming the project and selecting a location, clicking “OK” will open up the “Add-in Wizard.” In the next step, there will be an opportunity to select to “Create an Add-in using Visual C#,” followed by a prompt to “Select an Application Host.” The developer can then select as many applications as he would as host applications for the Add-in; In this case, only PowerPoint is needed. After entering the “Name” and “Description” for the Add-in, an “Add-in Options” page will enable the developer to specify loading and availability options. After providing a summary of the Add-in’s options, the developer can click “Finish” and a Solution will be created which includes two projects, one in which the Addin’s skeleton, “Connect.cs,” resides, and another which is the Add-in’s “Setup” project.

4.2.2 Graphical User Interface

I. Controls

The controls for the graphical user interface are added as an extra IAT ToolBar in PowerPoint within the function “OnStartupComplete” within “Connect.cs.” The functionalities for each of the buttons on the toolbar also have an equivalent affordance within one of the drop down menus at the top of the PowerPoint application window.

When adding custom images to toolbar items within Microsoft Office applications, one method is to employ “AxHost,” a class used by the “AxImp” tool to wrap “ActiveX” controls and expose them as “Windows Forms” controls. The following is a sample procedure that can be used to create a custom button image:

- Use the “GetIPictureDispFromPicture” method to take an “Image” object and converts it to an “IPictureDisp” object, the type of the “Picture” property exposed by the “CommandBarButton” class
- Add the “stdole” Interop Assembly to the list of references since “IPictureDisp” is an interface defined in the “stdole” type library and Interop Assembly
- Create .resX file for use for button images

In order to create a .resX file, a separate resource editor, called “ResourceEditor,” was created as a Visual C# project and used to generate the necessary images to use on the IAT toolbar buttons.

II. Inserting an Exercise Slide

When the “Insert Exercise Slide” button, marked with a “Red Pen” icon, , on the IAT

toolbar or under the “Insert” menu is clicked, the current slide index is retrieved and used to add a new slide with the slide index incremented. A new TextBox shape is then added to the new slide and given the name “QuestionText.” The text for “QuestionText” is set to “Click to add Question Text for Exercise” and the number of the current exercise. A second TextBox, named “AnswerInstructionText” is also created to give the instructions “Click on icon to add an Answer Box with mode “Keyboard” or “Pen”.” All this is done using the “Microsoft Office PowerPoint 2003 Visual Basic for Applications (VBA) Language Reference.”

Finally, a new object of type “KeyboardPen” is also generated and displayed with icons indicating interactions modes “Keyboard” and “Pen,” and the tag “Exercise” containing a serialized empty exercise is also created and added to the slide. This is in preparation for all future editing performed on the slide, since changes will always be made permanent by unserializing, editing, and reserializing the “Exercise” tag. A screenshot of the active slide is shown in **Figure 1**.

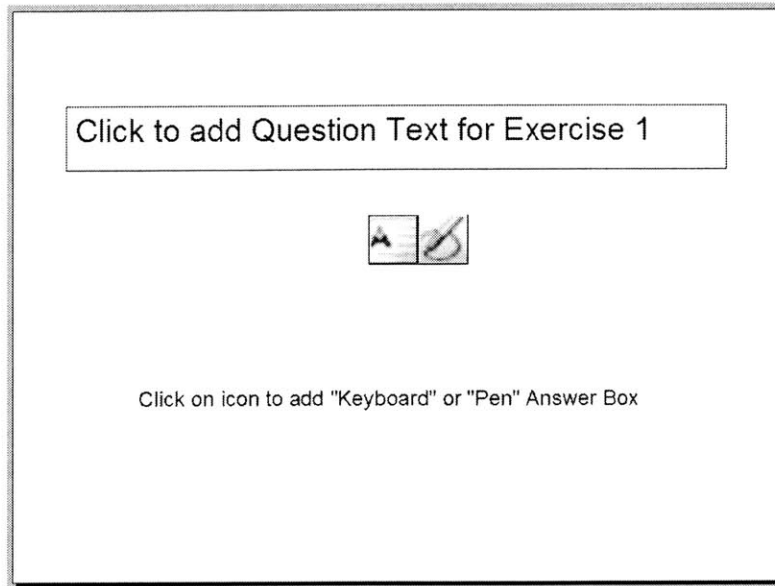




Figure 1. View of active slide after user clicks on a button to insert Exercise Slide.

- **The “KeyboardPen” Class**

Originally, the intention was to embed the icons for Keyboard and Pen within the PowerPoint slide itself. However, the VBA Language Reference was insufficiently helpful to successfully perform all aspects of the embedding. As a result, the “KeyboardPen” class evolved as an alternative built in Visual C#. The class consists of just two button icons, each with its own handler, residing in a frameless window. The handler for the “Pen” icon, , currently displays a MessageBox with the text “The PEN mode of interaction has not yet been implemented.” The handler for the “Keyboard” icon, , deletes the “AnswerInstructionText” TextBox, and inserted a new rounded-rectangular shape named “AnswerBox” with the text “Keyboard Answer Box” where the user can enter the primary or first answer to this exercise. The “KeyboardPen”

class also creates and displays an object of type “ExpectedType.” A screenshot of the active slide is shown in **Figure 2**.

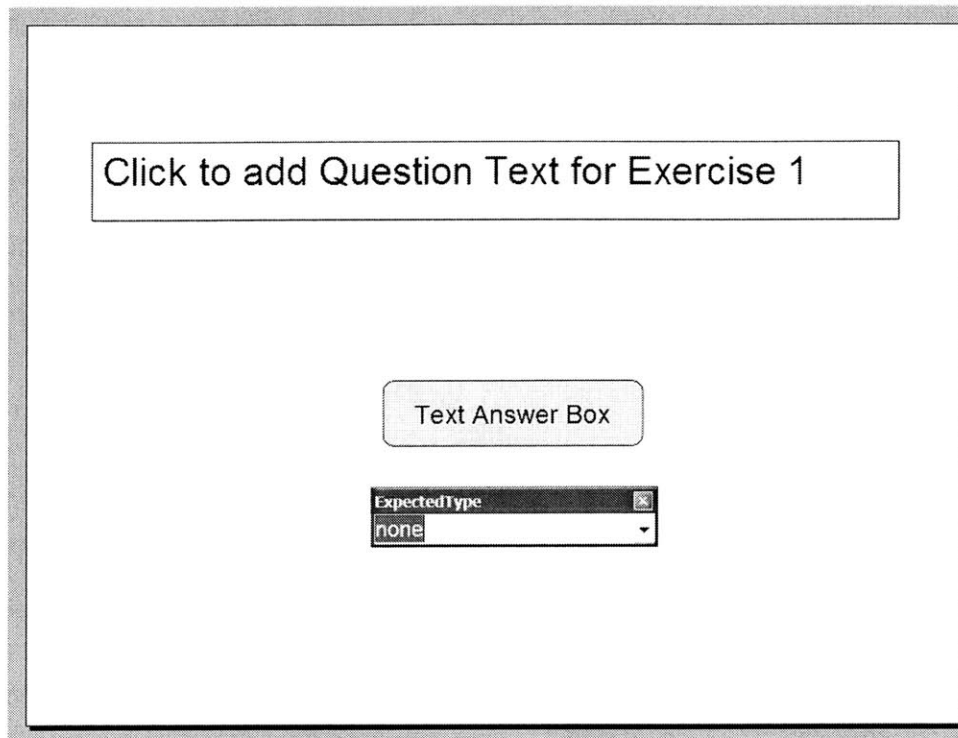


Figure 2: View of active slide after clicking on the “Keyboard” interaction icon.

- **The “ExpectedType” Class**

The “ExpectedType” class is a windows form that consists of just a single ComboBox containing all the possible answer types: None, Number, Text, Sequence, Set, and Scheme-Expression. The ExpectedType form was originally created with no border and a white background to give the effect of being embedded within the PowerPoint slide. However, the FormBorderStyle

was changed to a FixedToolWindow for the user's ease of moving it around on screen, and closing the form when desired. When a user selects an item in the ComboBox, the "type_SelectedIndexChanged" handler checks to see if the chosen item was "SchemeExpression." If not, the handler does nothing; Otherwise, an instance of the "SchemeEx" form is created and displayed.

- **Provision for Answers Containing Scheme Code**

As the target class for the first round of system testing for CLP is MIT's introductory course in Computer Science, *6.001 Structure and Interpretation of Computer Programs*, the IAT is implemented to support the creation of exercises with an expected answer type of "SchemeExpression."

The "SchemeEx" Class

The "SchemeEx" class is the form that allows a user to create a new answer of type "Scheme-Expression." When it is first displayed, the user sees only a form containing a single GroupBox labeled with "Answer #1" containing TextBoxes for entering "Keywords IN" and "Example Scheme Answer." There are also two CheckBoxes labeled "Match" and "Run" indicating the method by which the answer should be tested, and also the buttons "Cancel" and "Commit Exercise." A screenshot is shown in **Figure 3**.

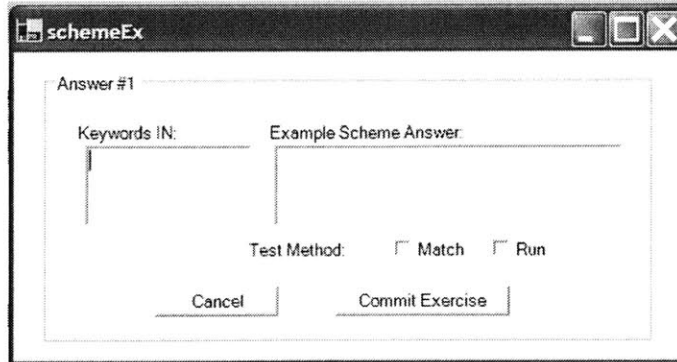


Figure 3. SchemeEx form when it is first displayed.

If the user clicks on “Cancel,” the window is closed with no other activity. If “CommitExercise” is clicked with neither test methods checked, the default is “Match.” If “Run” is checked at any point, new fields for input entry will appear. First, a TextBox for “Text Code” which a CheckBox labeled “Use Example Scheme Answer as Test Code.” When this is checked, the text “<Use Example Scheme Answer as Test Code>” will appear in the “Text Code” TextBox. A second TextBox allowed the user to enter the “Pre-execution Environment,” and underneath are fields for entering “Test Cases.” When first displayed, only fields for a single test case appears, along with buttons labeled “Add New Case,” “Delete Selected,” “Commit Exercise,” and “Cancel.” A screenshot is shown in **Figure 4.**

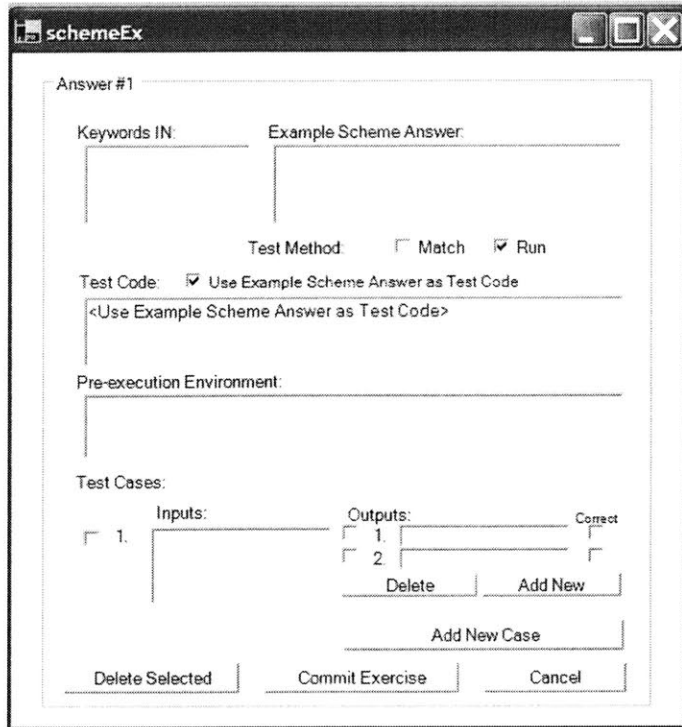


Figure 4. SchemeEx window after user checks “Run” and “Use Example Scheme Answer as Test Code”

Dynamically Displaying Fields for the “Run” Method

To enable information to be displayed on a need-basis in this way, the entire “SchemeEx” form minus the end buttons is first laid out and created in Windows Forms. During initialization, SchemeEx checks a “memo” variable to ensure that SchemeEx is being called in the current context from “ExpectedType” (as it will also be callable elsewhere). If so, “SchemeEx” assumes the current exercise is being attached to a slide for the first time. From here, SchemeEx creates a set of `system.Drawing.Rectangle` objects that are used to store the locations and sizes of all the relevant items. (A set of

rectangles are also created which represent the current location of the bottommost set of controls corresponding to the case with the highest number, as these will change as new cases are added.) After all necessary locations and sizes have been stored, all items below the Test Methods CheckBoxes are also removed from the GroupBox. The buttons for “Cancel” and “Commit Exercise” are added and the GroupBox and window are reset to the appropriate size.

Test Case Entry: Two Levels of Dynamic Tables

Once the “Run” CheckBox has been checked, the fields for test case entry are among the controls that appear. In order for them to function as desired, these controls are set up as a dynamic table, in this case, on two levels.

❖ Level One

A single case consists of a CheckBox, a Label with the appropriate row number, a TextBox for entering parameters, a set of outputs with all its components, and buttons for editing the outputs. When “Run” is first checked, space for one case is displayed as the default.

▪ *Add New Case*

On the first level, an entire new row corresponding to a new case needed to be added when the user clicks on the “Add New Case” button. To set up for this maneuver, a set of rectangles are created,

storing the locations and sizes corresponding to the controls that are a part of the current, or bottommost, case. When the “Add New Case” button is clicked, an integer called “step” is calculated and passed on to a function called “CalculateNewLS”; “step” is calculated based on the size of the previous entry, and “CalculateNewLS” resets the locations of the reference Rectangles to the appropriate values for the new controls to be added. Next, the “AddBlankRow” function is used to create a new set of controls using the updated reference Rectangles. The end buttons are removed, the new set of controls are added, and the “AddNewEndButtons” function is then used to restore the “Add New Case,” “Delete Selected,” “Commit Exercise,” and “Cancel” buttons, as well as to reset the GroupBox and window again to the appropriate size. The tags in all the controls that are part of a test case are set to an integer corresponding to the row number of the test case.

- ***Delete Selected***

The CheckBox to the left of each case is used to remove the case from the display during editing. When the user checks any of the boxes and clicks on the “Delete Selected” button, the “buttonDelete_Click” handler first creates a list of all the tag values that has been selected for deletion. Next, the handler goes through

all the test case rows and first stores all the test cases in an ArrayList as they appear on the screen. Next, the handler runs the selected list of tags for rows to delete through a standard Quicksort function to enforce numerical ordering. The handler then runs through the list containing the original cases on the screen in reverse order and utilizes the quicksorted list to remove all the cases selected for deletion in backwards numerical order.

With the new set of cases established, the handler then calls the “DeleteCases” function which removes all end buttons as well as the controls belonging to all cases from the GroupBox. The reference rectangles are now reset to their original positions, and the handler then loops through the new set of cases, calculating a “newStep” and calling the “DrawCaseTable” function each time which takes the row number of the case in consideration, creates and initializes all the controls for the case, and adds them to the GroupBox. When this is done, “AddNewEndButtons” is called again to reframe the window.

❖ **Level Two**

When a new case is added, space for entering two different sets of outputs are automatically displayed. A single output consists of a CheckBox, a Label containing the row number of the output, a TextBox for the output text, and a CheckBox indicating whether the output is

considered correct or not. In order for the number of outputs to be easily editable as is the case for the number of cases, a second level of dynamic tabling needed to be embedded within the first.

- ***Add New Output***

When the user clicks on the “Add New” button for the outputs of any case, the handler calls the “AddBlankOutput” function which acts very much like the handler for deleting cases. It first stores in an ArrayList what the current set of cases look like on the screen, adds an extra blank output where the user has indicated, resets the reference Rectangles to their original positions, and redraws the table with the new number of outputs for each case. Within the “DrawCaseTable” function, a *for* loop is used to create and display all the controls for a given set of outputs. A new function “CalculateNewOT” is used to relocate the reference Rectangles for the bottommost output row. This ensures that the table will display correctly with any variable number of outputs per case, and all other control displays can be fitted to take into account the variable nature of the amount of space needed for all outputs to properly display for each case.

- ***Delete Output***

Deleting an output works in the same way as deleting a case from

the user's perspective. When the user checks the CheckBoxes in front of unwanted outputs and clicks on the appropriate "Delete" button for that case, the "DeleteOutput" function again creates a snapshot of the screen. This time, it quicksorts the outputs selected for deletion, removes them from the current set of outputs for the appropriate case, and calls the "DeleteCases" button. From there, a similar procedure is followed by which the reference Rectangles will be reset, and calls to CalculateNewLS, DrawCaseTable, and CalculateNewOT, will be made, finishing off with a call to the AddNewEndButtons function for reframing.

The "Commit Exercise" Button


After entering all the desired information, the user will click on the "Commit Exercise" button. When this happens, the handler calls the "Commit" function which will take down the cases as they appear on the screen one last time, and also record the inputs for keywords-in, example scheme answer, test code, and pre-execution environment from the user; The expectedType for this answer is automatically set to "SchemeExpression." The "Commit" function then checks the "memo" variable again to see whether "SchemeEx" is being called from "ExpectedType." If so, "Commit" also records the question text from the appropriate shape on the current slide, creates a label for the exercise if it does not already exist, closes the "ExpectedType" form, resizes the current "AnswerBox" shape and sets its text to null, and also creates a "Comment" tag

which tags the box with the expected type, answer text, and date of creation. In this case, the answer text would be set to “<SchemeExpression>.” The exercise is also serialized and placed in the “Exercise” tag of the current slide.

The “Cancel” Button

When the user clicks on the “Cancel” button, the “SchemeEx” window closes, and no other change occurs.

III. Committing an Exercise

If the user chooses any type other than “SchemeExpression” from ComboBox that is a part of the “ExpectedType” instance, they can simply enter the text of their answer in the answer box provided and click on the “Commit Exercise” button in the toolbar, marked with a “BullsEye” icon (), or from the “Edit” menu. When this happens, a process occurs that is similar to that in the handler of the “Commit Exercise” button under “SchemeEx.” The expectedType for the first answer is set to the text in the ComboBox. If no answer has been entered in the AnswerBox shape or if the QuestionText box is empty, a MessageBox is displayed with the message “No answer has been entered,” or “The Question Text box is blank.” If neither of these conditions are true, then the “ExpectedType” window is closed, the AnswerBox shape is resized with the text set to null, and a Comment tag containing the expected type, the answer text, and the date of creation is created and placed in the lower right corner of the AnswerBox. Just as in the case with a scheme answer, a label containing the number of the exercise is created in the upper left corner of the slide, and the new exercise containing the current question text, answer, and

student answer area is serialized and placed in the “Exercise” tag of the current slide. A screenshot of the active slide is shown in **Figure 5**.

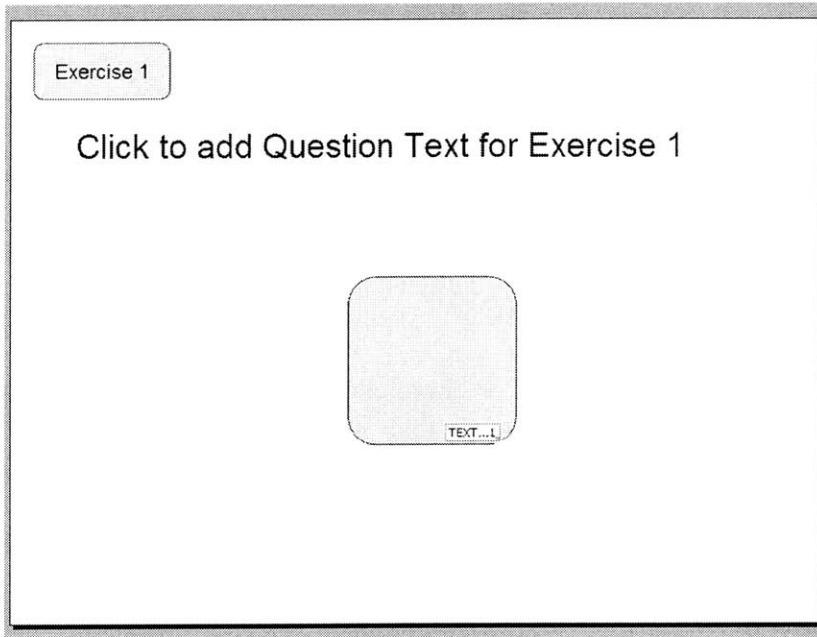



Figure 5. View of active slide after user clicks “Commit Exercise” from IAT ToolBar or the “Edit” menu.

IV. Displaying List of Exercises, Exercise Details, and Editing Exercise Answers

At any point during the creation of the slide deck, the user can click on the “Display List of Exercises” button from the IAT toolbar, , or the “View” menu, an instance of the “ExerciseList” class will appear bordering the left edge of the screen, displaying a list of all the exercises currently in the slide deck by their exercise number, slide number, and the first portion of their question text if it exists.

- **The Exercises ListBox**

The form for the exercise listing contains a single GroupBox labeled with the format of display: “Exercise # ~ Slide # ~ Question Text.” Inside the GroupBox is a Label containing the instructions “Click to Display Exercise Details,” and a ListBox whose background color has been set to a light grey to indicate an affordance for clicking. When the user clicks on any of the exercises listed, a new window appears containing the details for this exercise, and also allows the exercise to be edited. A screenshot of the “ExerciseList” window is shown in

Figure 6.

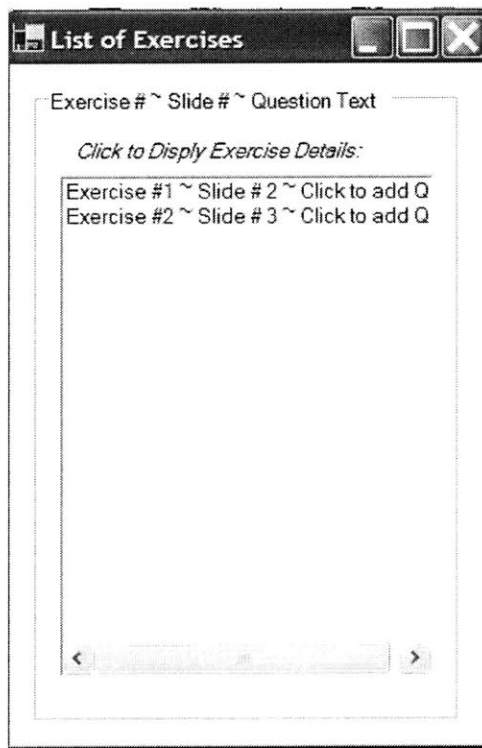



Figure 6. View of the ExerciseList window after user clicks the “Display List of Exercises” button ().

- **The Exercise Details Window**

The exercise details window also contains a single GroupBox, labeled with the exercise number. Inside the GroupBox, a TextBox displays the question text and another allows any notes for the exercise to be entered. Underneath the question text, the expected type and Interaction mode are displayed, both in ComboBoxes and editable. The current answers for the exercise are then displayed in a dynamic table similar to the one in SchemeEx. Below the answers are the buttons “Add New Answer,” “Delete Selected,” “Apply Changes,” “OK,” and “Cancel.” A screenshot of the ExerciseDetails window is shown in **Figure 7**.

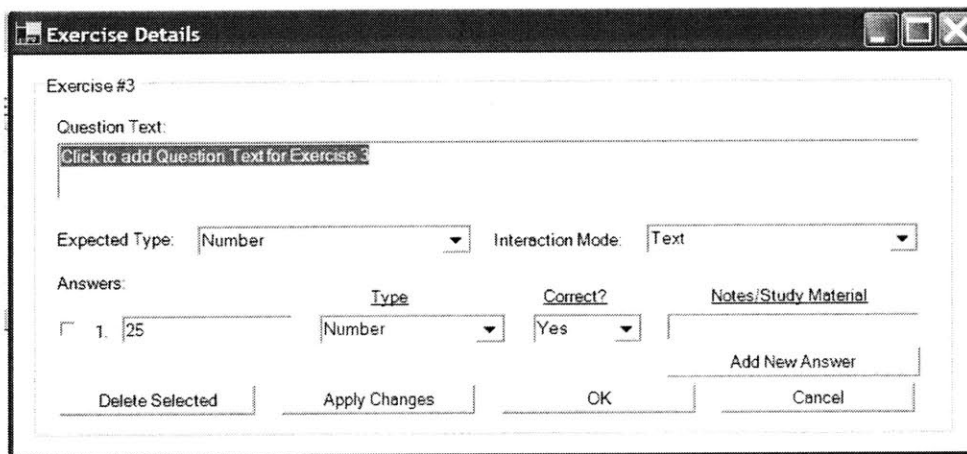


Figure 7. View of ExerciseDetails window.

- **The “Add New Answer” Button and the Dynamic Table**

The dynamic table for answer display works the same way as the dynamic table in SchemeEx, but has only a single level. Each row for an answer consists of the following controls: A CheckBox for deletion, a label containing the answer number,

a TextBox for answerText, a ComboBox for answer type, a comboBox indicating whether the answer is correct, incorrect, or neither, and a TextBox for the instructor to enter any notes or study materials associated with each answer. Each time the “Add New Answer” button is clicked, handler calls the “CalculateNewLS” function which updates the reference Rectangles, and calls the “AddBlankRow” function which removes all the buttons from the form, creates a new set of controls for a blank row, adds them to the GroupBox, and calls the “AddNewEndButtons” function to have the “Add New Answer,” “DeleteSelected,” “ApplyChanges,” “OK,” and “Cancel” buttons readded to the GroupBox and the GroupBox and window resized. The tags for all the controls associated with an answer are set to their corresponding answer number. The handler also enables the “Apply Changes” button to ensure that the user has the opportunity to save changes.

- **Exercise Details for Scheme Answer**

If the exercise for display contains any answers of type “SchemeExpression,” then an instance of the “SchemeAnswersList” class will be created and displayed when “ExerciseDetails” is initialized. This window is aligned to the left of the ExerciseDetails window, and looks very much like the ExerciseList form and lists all the answers by answer number for the exercise of type “SchemeExpression.” When the user clicks on any listed answer, an instance of “SchemeExDetails” is created and displayed beneath the SchemeAnswersList window. **Figure 8** shows a partial shot of the screen with all three windows opened.

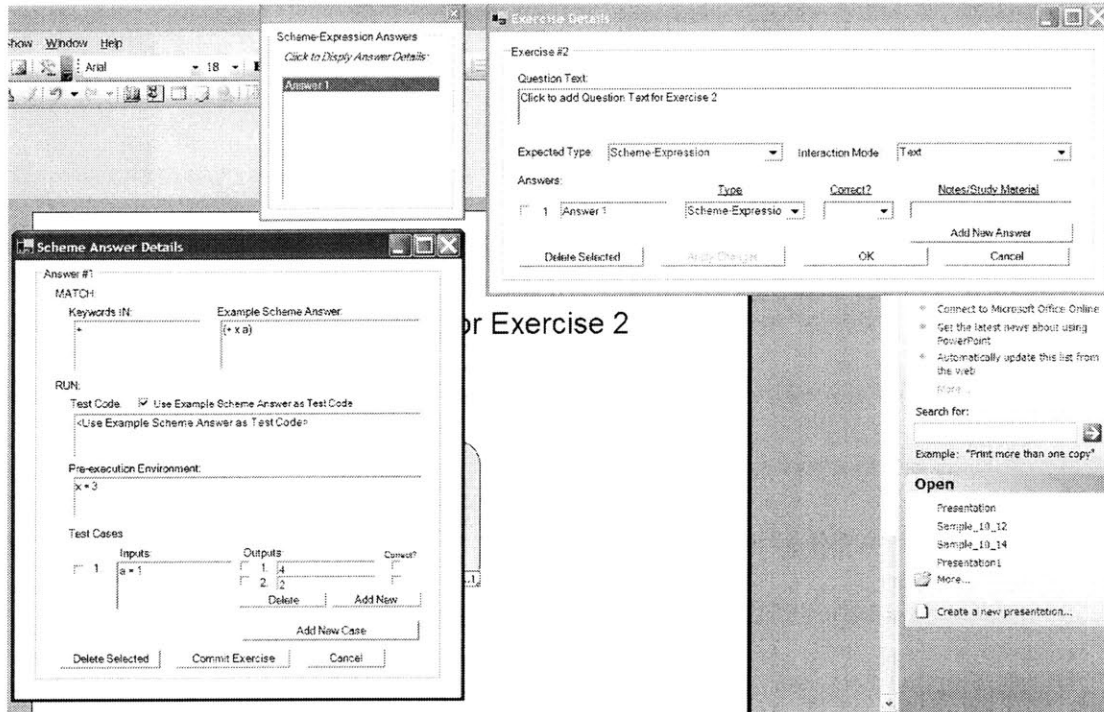


Figure 8. Screenshot showing placement of ExerciseDetails, SchemeAnswersList, and SchemeExDetails windows.

- **The “SchemeExDetails” Class**

The “SchemeExDetails” class is used to display and edit the information from a scheme answer that has already been created, and is instantiated only through SchemeAnswersList’s “listBoxSchemeAnswers_SelectedIndexChanged” handler. The window consists of a GroupBox labeled by the answer number. Within the GroupBox, under the label “MATCH,” are TextBoxes for displaying keywords-in and the example scheme answer. Under the label “RUN” are TextBoxes containing the text code and the pre-execution environment, along with the CheckBox for indicating whether to use the example scheme answer as the test code. Below the

pre-execution environment, the test cases committed for this scheme answer are displayed within a two layer dynamic table like the one used in SchemeEx. Similarly, the buttons “Add New Case,” “Delete Selected,” “Commit Exercise,” and “Cancel” appear on the bottom. This form can be use in the same way as SchemeEx to make any desired changes to the scheme answer in display. Clicking on the “Cancel” button closes the SchemeExDetails window without taking any action, and clicking on the “Commit Exercise” button triggers a handler which records the changes, unserializes the exercise on the slide in question, modifies the scheme answer in question, reserializes the exercise and resets the “Exercise” tag on the slide. The commit handler also closes the current SchemeAnswersList window and redisplay one containing the updated scheme answers. A closeup view of the SchemeExDetails window is shown in **Figure 9**.

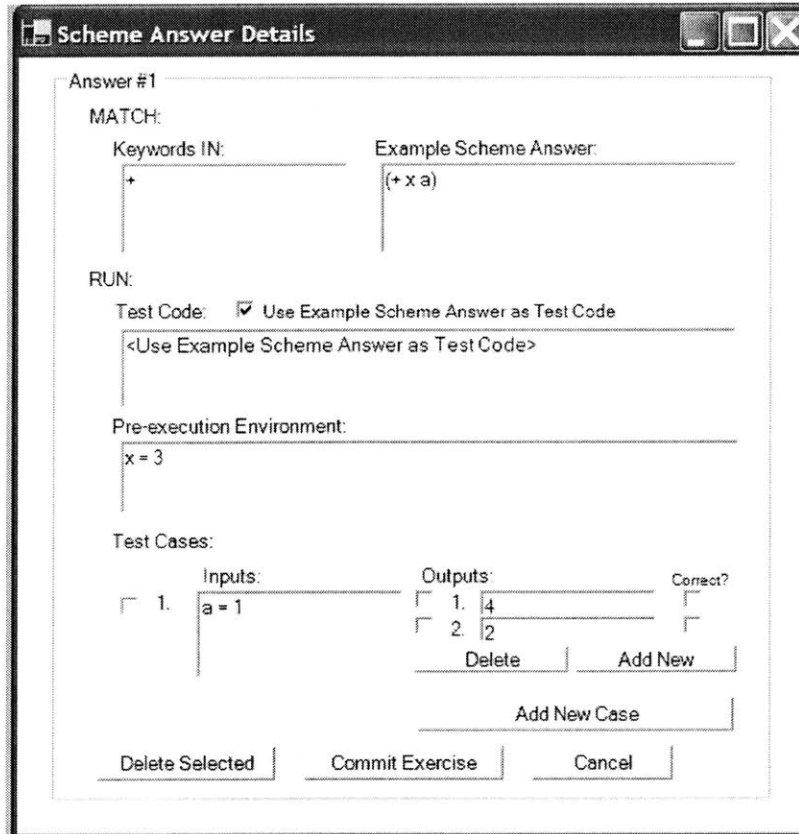


Figure 9. Closeup view of SchemeExDetails window.

- **Adding a new Scheme Answer**

If the user decides to add a new answer and selects “SchemeExpression as the type, a SchemeEx form is displayed underneath the ExerciseDetails window. The button “Apply Changes” on the ExerciseDetails form is disabled, as clicking “Commit Exercise” in SchemeEx or SchemeExDetails would be equivalent to “Apply Changes” for answers of type “SchemeExpression.” The memo variable is used to distinguish this instance of SchemeEx from one called from the “ExpectedType” class. In the end, when the user clicks on the “Commit Exercise” button after filling in all the information desired, the handler updates the “Exercise” tag accordingly

for the appropriate slide, the existing SchemeAnswersList window is closed along with any SchemeExDetails window that might be open, and a new SchemeAnswersList window is displayed which reflects the addition of the new scheme answer. The ExerciseDetails window also needs to be reestablished to reflect the changes, again as the commit is equivalent to “Apply Changes.” The developer should be careful to pass both the SchemeAnswersList and ExerciseDetails instances to SchemeEx and SchemeExDetails to allow such control. Note that in the case of the exercise being of expected type “SchemeExpression,” the “Apply Changes” button is still relevant when question text, expected type, if the answer is correct, or notes and study material is modified. It would also be needed if the user chooses to enter an answer of a type other than “SchemeExpression” when the exercise is of expected type “SchemeExpression,” although this situation should arise rarely. A partial view of the screen with the new SchemeExpression type answer being added from ExerciseDetails is shown in **Figure 10.**

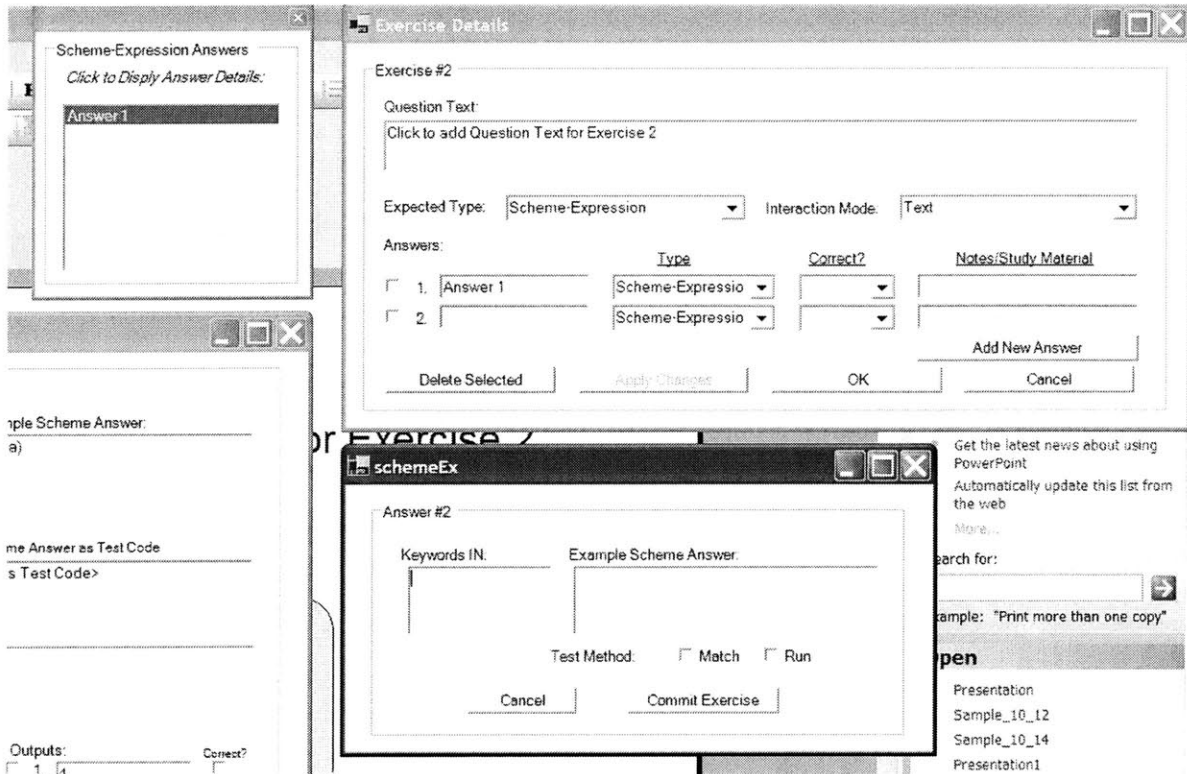


Figure 10. A partial view of the screen when the user tries to add a new “SchemeExpression” type answer from the ExerciseDetails window.

- **The “Apply Changes” Button**

When the user clicks on the “Apply Changes” button after having completed all editing within the ExerciseDetails form, the handler records each of the fields on the form, being careful to distinguish between adding an answer text versus a RunnableAnswer depending on the type chosen for the answer in each row. Next, the new exercise is serialized and added to the “Exercise” tag of the appropriate slide. Finally, the “Apply Changes” button is disabled when all actions are completed.


- **The “OK” Button**

Clicking on the “OK” button performs the same function as the “Apply Changes” button but, in addition, also closes the ExerciseDetails window and the SchemeAnswersList window if they exist.

- **The “Cancel” Button**

Clicking on the “Cancel” button closes the ExerciseDetails window and SchemeAnswersList window if it exists without taking any action.

V. Saving Exercises to the Database

When the user has created his slide deck, clicking on the “Save Exercises to Database” button, , will store the created exercises to the database, making them available to other CLP modules. This handler loops through each of the slides in the slide deck, and first unserializes the exercise in the “Exercise” tag if one exists. This unserialized “ExerciseIAT” object is then used to generate “Exercise” objects that are in turn saved to the database using the “SaveExercise” of an instantiated “CDatabaseExport” object that has been initialized. A database ID of type “int” is returned as a result and can be used at any point to retrieve the exercise from the database if desired.

4.2.3 CLP-Wide Classes

There are several classes that needed to be created in the construction of the IAT as part of a larger relationship of classes representing fields to be stored in the CLP database repository. In addition to being essential for certain modules to function, this repository

will also become the basis of future extensions such as the construction of an intelligent tutor-type system. These classes are used within the IAT during database store and also export to .CSD to generate objects compatible for integration with CLP. These classes are “AnswerMetadata,” “Case,” “Exercise,” “InstructorAnswer,” “Question,” and “RunnableAnswer.” (The other classes currently in the larger CLP-Wide set of classes are “Answer,” “BaseClass,” “Lecture,” “Student,” and “StudentAnswer.”) All of these classes fall under the “CLPClasses” project.

I. The “AnswerMetadata” Class

The “AnswerMetadata” class is currently used to store notes input by the instructor for each answer within an exercise. This information is stored in the “description” member.

Its Private Members are:

```
private int parentAnswerId;  
private string description;  
private string misunderstoodConcept;  
private string studyMaterial;
```

which all have corresponding Properties.

“AnswerMetadata” must be marked with a “[Serializable]” tag in preparation for serialization.

II. The “Exercise” Class

The “Exercise” class is the basis of all exercises created by an instructor when it is passed onto Classroom Presenter in a .CSD file for use by other modules of the CLP, and saved

into the database repository.

Its Private Members are:

```
private int exerciseNumber;  
private int slideNumber;  
private Question question;  
private ExpectedType expectedType;  
private InteractionMode interactionMode;  
private ArrayList instructorAnswers;  
private int parentLectureId;  
private string notes;  
private Rectangle studentAnswerArea;  
private AnswerType answerType;
```

which all have corresponding Properties.

“Exercise” must be marked with a “[Serializable]” tag in preparation for serialization. In addition, it must also include the following tags:

```
[System.Xml.Serialization.XmlInclude(typeof(InstructorAnswer))]  
[System.Xml.Serialization.XmlInclude(typeof(RunnableAnswer))]  
[System.Xml.Serialization.XmlInclude(typeof(Case))]  
[System.Xml.Serialization.XmlInclude(typeof(Answer))]  
[System.Xml.Serialization.XmlInclude(typeof(BaseClass))]  
[System.Xml.Serialization.XmlInclude(typeof(Question))]  
[System.Xml.Serialization.XmlInclude(typeof(AnswerMetadata))]  
[System.Xml.Serialization.XmlInclude(typeof(AnswerType))]
```



```
[System.Xml.Serialization.XmlInclude(typeof(ExpectedType))]
```

```
[System.Xml.Serialization.XmlInclude(typeof(IfCorrect))]
```

```
[System.Xml.Serialization.XmlInclude(typeof(InteractionMode))]
```

This will allow the internal components of its instances to be serialized as part the instance.

III. The “InstructorAnswer” Class

The “InstructorAnswer” class extends “Answer,” and is the basis of every answer created by an instructor for an exercise when being passed on to a .CSD file or exported to a database.

Its Private Members are:

```
private string description;
```

```
private string ifCorrect;
```

```
private AnswerMetadata answerMetadata;
```

```
private int number;
```

which all have corresponding Properties.

“InstructorAnswer” must be marked with a “[Serializable]” tag in preparation for serialization. In addition, it must also include the following tags:

```
[System.Xml.Serialization.XmlInclude(typeof(Answer))]
```

```
[System.Xml.Serialization.XmlInclude(typeof(IfCorrect))]
```

```
[System.Xml.Serialization.XmlInclude(typeof(AnswerMetadata))]
```

This will allow the internal components of its instances to be serialized as part the instance.

IV. The “Question” Class

The “Question” class extends “BaseClass,” and is used by the IAT to store question text for each exercise when being passed on to a .CSD file or exported to a database.

Its Private Members are:

```
private int parentExerciseId;  
  
private String questionText;
```

which all have corresponding Properties.

“Question” must be marked with a “[Serializable]” tag in preparation for serialization. In addition, it must also include the following tag:

```
[System.Xml.Serialization.XmlInclude(typeof(BaseClass))]
```

This will allow the internal components of its instances to be serialized as part the instance.

V. The “RunnableAnswer” Class

The “RunnableAnswer” class extends “InstructorAnswer,” and is used to store information for all answers of type “SchemeExpression” created by the instructor when being passed on to a .CSD file or exported to a database.

Its Private Members are:

```
private string keyIn;  
  
private string exampleCode;  
  
private string testCode;  
  
private string preEx;  
  
private ArrayList cases;
```

which all have corresponding Properties.

“RunnableAnswer” must be marked with a “[Serializable]” tag in preparation for

serialization. In addition, it must also include the following tag:

```
[System.Xml.Serialization.XmlInclude(typeof(InstructorAnswer))]
```

This will allow the internal components of its instances to be serialized as part the instance.

VI. The “TestCase” Class

The “TestCase” class is used to store information about each case in a ShemeAnswer when passed on to the .CSD file or exported to the database as part of an exercise.

Its Private Members are:

```
private int parentAnswerId;
```

```
private string parameters;
```

```
private ArrayList outputs;
```

```
private ArrayList ifCorrect;
```

which all have corresponding Properties.

“TestCase” must be marked with a “[Serializable]” tag in preparation for serialization. In addition, it must also include the following tags:

```
[System.Xml.Serialization.XmlInclude(typeof(BaseClass))]
```

This will allow the internal components of its instances to be serialized as part the instance.

4.2.4 IAT-Wide Classes

Certain IAT-wide classes with more efficient formatting had also been created for use before storing to database and exporting to .CSD . These are “ExerciseIAT.cs,” “InstructorAnswerIAT.cs,” “RunnableAnswerIAT.cs,” and “TestCaseIAT.cs.” All of these classes fall under the “IATClasses” project.

I. The “ExerciseIAT” Class

The “Exercise” class is the basis of all exercises when first created by an instructor. It is then used to generate the “Exercise” object passed onto Classroom Presenter in a .CSD file, and for saving to the database repository.

Its Private Members are:

```
private int exerciseNumber;  
private int slideNumber;  
private ExpectedType expectedType;  
private InteractionMode interactionMode;  
private ArrayList instructorAnswers;  
private int parentLectureId;  
private string notes;  
private Rectangle studentAnswerArea;
```

which all have corresponding Properties.

“Exercise” must be marked with a “[Serializable]” tag in preparation for serialization. In addition, it must also include the following tags:

```
[System.Xml.Serialization.XmlInclude(typeof(InstructorAnswerIAT))]  
[System.Xml.Serialization.XmlInclude(typeof(RunnableAnswerIAT))]  
[System.Xml.Serialization.XmlInclude(typeof(TestCaseIAT))]
```

This will allow the internal components of its instances to be serialized as part the instance.

II. The “InstructorAnswerIAT” Class

The “InstructorAnswerIAT” class the basis of every answer when first created by an instructor for an exercise. It is then used to generate “InstructorAnswer” objects to be passed onto a .CSD file or exported to a database as part of an exercise.

Its Private Members are:

```
private string description;  
  
private string ifCorrect;  
  
private string answerMetadata;  
  
private int number;  
  
private string answerText;  
  
private string expectedType;
```

which all have corresponding Properties.

“InstructorAnswer” must be marked with a “[Serializable]” tag in preparation for serialization.

III. The “RunnableAnswerIAT” Class

The “RunnableAnswerIAT” class extends “InstructorAnswerIAT,” and is used to store information for all answers of type “SchemeExpression” when first created by the instructor. It is then used to generate “RunnableAnswer” objects when appropriate as they are passed on to a .CSD file or exported to a database as part of an exercise.

Its Private Members are:

```
private string keyIn;  
  
private string exampleCode;
```

```
private string testCode;  
private string preEx;  
private ArrayList cases;
```

which all have corresponding Properties.

“RunnableAnswer” must be marked with a “[Serializable]” tag in preparation for serialization.

IV. The “TestCaseIAT” Class

The “TestCaseIAT” class is used to store information about each case in a SchemeAnswer when first created. When “TestCase” is instantiated, the constructor adds two empty string objects to the output member, and two default “NO” strings to the ifCorrect member.

Its Private Members are:

```
private string parameters;  
private ArrayList outputs;  
private ArrayList ifCorrect;
```

which all have corresponding Properties.

“TestCaseIAT” must be marked with a “[Serializable]” tag in preparation for serialization.

4.2.5 Integration with “InstructorModeAddin”

In order for the IAT add-in to be integrated with the InstructorModeAddin, the connect.cs file from each were merged. Changes were also made to the “SlideViewer.Slide” class, as well as the “PPTDeckBuilder.PPTSlideLoader” class.

I. Modifications to the “SlideViewer.Slide” Class

An extra private member of type “System.Collections.ArrayList” called slideExercises is added to the “SlideViewer.Slide” class as a vehicle for passing all relevant information from instructor inputs to the .CSD version of the slide deck for use by another CLP module.

II. Modifications to the “PPTDeckBuilder.PPTSlideLoader” Class

- Provisions were made within the “PPTDeckBuilder.PPTSlideLoader” class for passing the relevant instructor input on to the exported .CSD file for use in Classroom Presenter. Within “PPTSlideLoader,” “BuildSlideDeck” is the function responsible for creating an array of “SlideViewer.Slide” objects for use in constructing the .CSD. An addition was made, so that upon export, “BuildSlideDeck” would go through each PowerPoint slide in the slide deck, unserialize all the exercises on each slide as “ExerciseIAT” object, generate appropriate “Exercise” object from them, and attach these to the corresponding “SlideViewer.Slide” through its private member and property “SlideExercises.”
- In addition, changes were also made to the public method “LoadSlidesFromPresentation” from “PPTSlideLoader.” Because the InstructorModeAddin has created functionality to easily switch back-and-forth between instructor and student view, it is important for the instructor to be able to modify the student answer area in one view and not have to repeat the process in the

other. At the same time, it is also important to have on display the answer text in each AnswerBox in instructor view only as a lecture-aid. Within PowerPoint, this is resolved by means of the Comment tags. For viewing in Classroom Presenter, a separate instructor answer box containing the answer text is created on-the-fly during export, and removed immediately following the process. This procedure occurs within the “LoadSlidesFromPresentation” function where, for each exercise slide, an “InstructorAnswerBox” shape containing the answer text, identical in size and location to the current “AnswerBox,” is created and set to “Instructor” mode only. Based on this information, the “BuildSlideDeck” function will then ensure that the answer text is added only to the slide image used for the “Instructor” view in .CSD. When the function has completed the export, the “InstructorAnswerBox” shapes are permanently removed. Thus, the process is completely transparent to the user.

III. Compatibility with Classroom Presenter

In generating a .CSD file, the InstructorModeAddin performs serialization to generate the correct format. Thus, when Classroom Presenter is ready to open a .CSD file for use, the file undergoes a process by which it is unserialized. This unserialization will be successfully performed as long as the developer is careful to update shared projects between the Presenter source code and the new joint InstructorModeAddin/IAT Addin which performs the original serialization in exporting to .CSD.

Section 5: Future Work

Currently, the top priority for extending the IAT is to make provision for creating exercise slides that contain multiple exercises, each with its own answer box. When this is possible, an instructor will be able to create slides that are more similar to the ones currently used for in-class exercises with multiple parts.

A basic plan for accomplishing this will involve either adding to the “Insert Exercise Slide” or replacing it with an “Insert Exercise” function. One of the first steps would be to design a scheme for allocation of space, and placement of exercises on the slide as they are added. Other questions to consider include whether to have question textboxes inserted automatically with an exercise, or to insert textboxes only and assume the same question for additional textboxes when a question text already exists. One possibility would be for exercises to be added always in a designated area on the slide, say the upper right corner, and have the user move them to the location of his choice; Another would be to create a selection of templates with different numbers and layouts of exercises on a slide and have the user select the type of exercise slide to insert. Then all the exercises can either be present already when the slide is inserted, or added one by one as desired in the preset locations as the user clicks on the “Insert Exercise.” The user is still free to move around objects and customize the slide as he sees fit.

Another aspect to consider is how much freedom to give to the user in committing exercises. Should all the exercises on a slide be required to be committed at once? Or perhaps only those whose answer boxes are selected? How should they be mapped to the ArrayList of exercises attached to the “Exercise” tag?

Other possibilities for future work include fine-tuning the SchemeExpression interfaces so that they can be used for code segments in other language, creating an interface for the user to generate custom answer types while building the slide deck, implementing the modified interface for using the “Pen” mode of interaction, and investigations into the PowerPoint API so that items like the “KeyboardPen” and “ExpectedType” type objects can be embedded directly into a slide rather than standing as separate windows. Investigations into attaching event handlers to PowerPoint shapes would also allow a more seamless compositional process where actions like committing and exercise can happen automatically, for example, as soon as the use enters an answer value.

Section 6: Results and Conclusion

Using the IAT, the user is currently able to successfully build a PowerPoint slide deck containing in-class exercises with expected answer types of “Number,” “Text,” “Sequence,” “Set,” “SchemeExpression,” or “None.” The slides can contain at most one exercise per slide, and the student answer area size and location can be set by the user for each exercise. The user is also able to see information for answer text, expected type, and date of creation, as well as a label containing exercise number in “Instructor” mode, while only the answer area is visible in the “Student” view of the slide. In addition, the user is able to export the .PPT slide deck into a .CSD file compatible with Classroom Presenter as long as the Presenter’s source code has been updated for compatibility with the IAT Addin. When opened in Classroom Presenter, the exported .CSD file will successfully display the appropriate information in both “Instructor” and “Student” mode. In addition, the user is

also able to save the exercises from his slide deck to the database repository for storage and integration with other CLP modules.

The IAT was able to successfully meet the design criterion of using Microsoft PowerPoint as a basic tool. It also laid the groundwork for leveraging past student mistakes and misunderstood concepts by providing a means for instructors to author multiple incorrect answers as well as correct ones and at the same time to attach any comments or notes to each. This criterion as well as the preparative work for an intelligent tutoring system have also been jump-started by the IAT's ability to successfully store all relevant instructor input into a database repository. While informal testing of the IAT for exercise authoring has been positive, both of the last two criteria, 1) enhancing support to instructors by facilitating the slide-generation process, and 2) enabling real-time feedback of student performance, have yet to be formally tested in a true academic setting. Nonetheless, it is hopeful that this implementation of the IAT would be is a part of an overall CLP system with the potential of being a critical turning point in the evolution of information technology as it applies to education in the 21st century.

References

Anderson, R., Anderson, R., Hoyer, C., Simon, B., Videon, F., Wolfman, S. (2005)

"Lecture Presentation from the Tablet PC," University of Washington

Anderson, R., Anderson, R., Simon, B. VanDeGrift, T., Wolfman, S., Yasuhara, K. (2004)

"Experiences With a Tablet PC-Based Lecture Presentation System," University of

Washington, University of Virginia

Anderson, R., Anderson, R. VanDeGrift, T, Wolfman, S., and Yasuhara, K. (2003)

"Promoting Interaction in Large Classes with Computer-Mediated Feedback, CSCL.

Anderson, R., Anderson, R. VanDeGrift, T, Wolfman, S., and Yasuhara, K. (2003)

"Interaction Patterns with a Classroom Feedback System: Making Time for Feedback", CSCL.

Anderson, R., Hoyer, C., Prince, C., Su, J., Videon, F., Wolfman, S. "Speech, Ink, and

Slides: The Interaction of Content Channels," Department of Computer Science and Engineering, University of Washington.

Bligh, D.A. (2000) *What's the use of lectures?* Jossey-Bass Publishers, San Francisco.

Bouillion, L.M., L.M. Gomez. "The Case of Considering Cultural Entailments and Genres

of Attachment in the Design of Educational Tehcnologies." *Smart Machines in Education: The Coming Revolution in Educational Technology*. Menlo Park, CA: AAI Press/MIT Press.

Brewer, C. A. (2004) Near Real-Time Assessment of Student Learning and Understanding in Biology Courses. *BioScience*, November 2004, Vol. 54 No. 11.

Brown, A. L. (1992) Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The Journal of the Learning Sciences*, 2: 141-178.

Cuban, L. (2001) *Oversold and Underused: Computers in the Classroom*. Cambridge, MA: Harvard University Press.

Davis, R. (2005), personal communication.

- Draper, SW. (2004) "From active learning to interactive teaching: Individual activity and interpersonal interaction," in *Teaching and Learning Symposium: Teaching Innovations*, The Hong Kong University of Science and Technology.
- Feltovich, P.J., R.L. Coulson, R.J. Spiro. "Learners' (Mis)Understanding of Important and Difficult Concepts: A Challenge to Smart Machines in Education." *Smart Machines in Education: The Coming Revolution in Educational Technology*. Menlo Park, CA: AAAI Press/MIT Press.
- Grimson, E. (2005), personal communication.
- Hinrichs, RJ. (2002) "Technology, Learning and Scholarship in the Early 21st Century," from Microsoft Corporation, Microsoft Research, Learning Science and Technology
- Higgins, E., L. Tatham. "Exploring the potential of Multiple-Choice Questions in Assessment." *Learning & Teaching in Action, Vol 2 Issue 1: Assessment*.
- Ho, B. (2001) "From using transparencies to using PowerPoint slides in the classroom", from AARE 2001 Conference Papers compiled by Peter L. Jeffery.
- Kadowec, J. (2005) "Using rapid feedback to enhance student learning in mechanics." From 35th ASEE/IEEE Frontiers in Education Conference.
- Koedinger, K. (2003) "Cognitive Tutors as Modeling Tools and Instructional Models." *Smart Machines in Education: The Coming Revolution in Educational Technology*. Menlo Park, CA: AAAI Press/MIT Press.
- Koile, K. (2005), personal communication.
- Laplante, Phillips and Wiesner, Peter. (2002). *Pedagogy for the Web-based technical education*.

<http://www.opencroquet.org/Site%20PDFs/Enabling%20Learning%202004.pdf>

Means, B., Blando, J., Olsen, K., Middleton, T., Morocco, C.C., Remz, A.R., Zorfass, J. (1993). "Using Technology to Support Education Reform." Study and report sponsored by the U.S. Department of Education, Office of Educational Research and Improvement, under Contract No. RR91172010.

Merrill, MD. "First Principles of Instruction." Submitted for publication to Educational Technology Research & Development.
<http://id2.usu.edu/Papers/5FirstPrinciples.PDF>

Papert, S. (1971) "Teaching Children to be Mathematicians vs. Teaching About Mathematics." To be published in the International Journal of Mathematical Education in Science and Technology (New York: John Wiley & Sons, 1972) and in Proceedings of 1970 CEMREL Conference on Algebra (Carbondale, Illinois: CEMREL, Spring, 1971).

Peiper, C., Warden, D., Chan, E., Capitanu, E., Kamin, S. "eFuzion: Development of a Pervasive Educational System," University of Illinois at Urbana, Champagne, Department of Computer Science

President's IT Advisory Committee. "Report to the President on Using Information Technology to Transform the Way We Learn."
<http://www.itrd.gov/pubs/pitac/pitac-tl-9feb01.pdf>

Ratto, M, Shapiro, RB, Truong, TM and Griswold, WG. (2003) "The Activeclass Project: Experiments in Encouraging Classroom Participation," in CSCL.

Regents of the University of Minnesota (2006) "Active learning with PowerPoint: Using PowerPoint to Facilitate Classroom Assessment Techniques." University of

Minnesota Center for Teaching and Learning Services.

Roschelle, J., W.R. Penuel, L. Abrahamson. (2004) "Classroom Response and Communication Systems: Research Review and Theory." Presented at the Annual Meetings of the American Educational Research Association, San Diego, CA, April 2004.

Satratzemi, M., Satratzemi, M., Dagdilelis, V. , Evagelidis, G. (2001) "A system for program visualization and problem-solving path assessment of novice programmers." from ACM 2001.

Schank, R., A. Neaman. (2003) "Motivation and Failure in Educational Simulation Design." *Smart Machines in Education: The Coming Revolution in Educational Technology*. Menlo Park, CA: AAAI Press/MIT Press.

Schwarm, S., and VanDeGrift, T. (2002) "Making Connections: Using Classroom Assessment to Elicit Students' Prior Knowledge and Construction of Concepts." In *Proceedings of the International Conference of the Learning Sciences*.

Simon, B., Anderson, R., Hoyer, C., and Su, J. (2004) "Preliminary Experiments with a Tablet PC Based System to Support Active Learning in Computer Science Courses," in *9th Annual Conference on Innovation and Technology in Computer Science Education (ITICE)*.

Simon, B., Anderson, R., Wolfman, S. (2003) "Activating Computer Architecture with Classroom Presenter." Workshop on Computer Architecture Education (WCAE), San Diego, USA. June 2003.

Virvou, M. (2000) "Involving Effectively Teachers and Students in the Life Cycle of an Intelligent Tutoring System." *Educational Technology & Society* 3(3) 2000.

- Warnick, B.R. (2003) "Review of *Smart Machines in Education Technology*." *Ed Rev*,
January 2003.
- Wilson, M., K. Scalise. (2004) "Using assessment to improve learning: The BEAR
Assessment System." University of California, Berkeley, March 2004.
- Woolf, B.P. (2003) "Growth and Maturity of Intelligent Tutoring Systems: A Status
Report." *Smart Machines in Education: The Coming Revolution in Educational
Technology*. Menlo Park, CA: AAAI Press/MIT Press.