The Design and Implementation of an Engine and Authoring Tool for Web-based Learn-by-doing Environments

Lin Qiu and Christopher K. Riesbeck
Department of Computer Science
Northwestern University
Evanston, Illinois 60201 USA
{qiu, riesbeck}@cs.northwestern.edu

Matthew R. Parsek
Department of Civil and Environmental Engineering
Northwestern University
Evanston, Illinois 60208 USA
m-parsek@northwestern.edu

Abstract: An important skill engineering students need to learn is how to solve problems by running experiments and using the results to make a sound diagnosis of the problem. This paper describes a software tool called Indie, which includes an authoring tool and a content-independent engine, for delivering web-based learn-by-doing environments where students can practice such skill in an authentic task. We focus on how the architecture of the tool makes it possible for teachers as non-programmers to customize the content of the learning environment as well as access and critique student activities in the learning environment. We present preliminary results showing that the Corrosion Investigator, a learning environment delivered by Indie, proves to be beneficial for both the students and teacher in a class.

Introduction

A significant amount of research has shown that learning is a process where the learner actively constructs knowledge and understanding instead of passively copying the external knowledge presented to them into their memory (e.g., Bransford, Goldman, & Vye, 1991; Brown, 1988; Chi et al., 1994; Graesser, Person, & Magliano, 1995). This theory has been widely accepted as the *constructivist* theory of learning and has become one of the dominant learning theories in education. Social theories of learning, meanwhile, show that learning is also a social process that often occurs through conversations as well as the construction of external artifacts (e.g., Collins, Brown, & Newman, 1989; Graesser, Person, & Magliano, 1995; Palincsar & Brown, 1984). All of this leads to a high demand for changes to the current drill-and-practice methodology of instruction. Within the trend, learn-by-doing has become a popular new paradigm of teaching. Learn-by-doing has the advantage of providing deeper engagement in the learning process, and superior retention and transfer of knowledge. Project-based learning (e.g., Blumenfeld et al., 1991), as one form of learn-by-doing, has largely been seen in schools where learning is centered around an investigation and development of artifacts and solutions to real problems. With this change in pedagogical strategy, the content of teaching has also been changed. In the engineering domain, teaching adaptive expertise (Bransford, Brown, & Cocking, 1999) becomes an important goal of the curriculum. Through critical thinking and scientific inquiry, students will learn not only engineering concepts but also the use of the conceptual knowledge as a set of tools (Brown, Collins, & Duguid, 1998). One important skill they need to learn is solving problems by running experiments and using the results to support or refute possible hypotheses. We call such activity "Investigate and Decide" (Dobson, 1998). With Investigate and Decide, students need to learn how to choose the correct experiments to run, interpret the relevance of the experimental results, and build a well-supported claim based on the evidence. A major obstacle with using such approach in school is that running experiments for realistic problems can be expensive, time-consuming, and even dangerous. For example, collecting water samples at a large processing plant as well as culturing bacteria can cost hundreds to thousands of dollars and take several weeks. Furthermore, students need coaching and feedback in order to use lab results efficiently and effectively. Using

role-play requires an instructor expert in the domain to generate detailed results and give feedback, which takes great effort and time.

One way to avoid these problems is to have students work in a computer-based learn-by-doing environment. A computer-based environment can provide immediate individualized feedback, web-based anytime anywhere accessible interface, and scaffolding for the learning process. Such interactive learning environments, however, are difficult and expensive to build. It is considerably harder for teachers, as non-programmers, to modify learning environments when they want to customize it for their courses. Such inflexibility inevitably hinders the learning environments being broadly used.

To solve these problems, we have built a new version of the software tool called Indie. Indie provides a content-independent software engine for delivering Investigate and Decide learning environments along with an authoring tool for constructing the content. Indie has been used to construct Corrosion Investigator (Qiu & Riesbeck, 2002), a learning environment for engineering undergraduates in the domain of biofilms. Corrosion Investigator is a computer-based version of a graduate-level engineering course project at Northwestern University. The course content addressed the biological and engineering concepts and biofilms. In class, students took the role of consultants helping a paper processing plant client diagnose recurring pipe corrosion. Students asked for background information about the company and ordered lab tests. The professor role-played characters in the scenario and generated information on demand, including fairly complex test results. At the end, students generated a report explaining their diagnoses with supporting evidence. The professor critiqued their arguments based on how well the claims were supported. All communications were done via email.

In the following, we will describe the architecture of Indie, how its implementation fulfills our design goal, and preliminary results from the use of Corrosion Investigator.

System Architecture

Indie is built based on the Goal-based Scenario (GBS) (Schank, 1994; Schank et al., 1993) framework for computer-based learn-by-doing environments. A GBS is designed to engage learners in active purposeful learning (Bransford, Brown, & Cocking, 1999). It begins by giving learners some large realistic problem that needs to be solved. With support from the learning environment, learners learn target skills and underlying concepts in pursuit of goals within a context rich environment. In Indie learning environments, students can run simulated experiments, construct reports arguing for different claims, and submit these reports for critiquing. Previous GBS systems (e.g., Bell, Bareiss, & Beckwith, 1994; Schank & Korcuska, 1996) were stand-alone systems. They do not provide effective ways for teachers to access student learning in the system. All feedback is generated by the software, which forces it to be pre-defined and limited. Additionally, they have been closed world systems. Once deployed, there is no easy way to add new operations, coaching, or critiquing at a later date in order to incorporate new developments in the domain or locally relevant content. Our new architecture, in contrast, allows teachers to access and critique students' work during runtime. By using Indie's authoring tool, it should be fairly easy for teachers to introduce new material or modify existing content in the system.

In order to be accessible, extensible, and reusable, our system needs to meet the following requirements:

- The architecture should be independent of the learning content.
- Authors should be able to make customized modules by making changes to the existing content without modifying a large part of the system.
- Authors should not need to specify the interactions in the system, since the learning environment generated by Indie is designed for doing Investigate and Decide activities.
- The learning environment and authoring tool should be general enough so that it can handle content in different domains.
- Teachers should be able to access and critique the students' work while students are still working with the system.
- The system should be client-server based with a web-based interface and minimum requirements on the client.

These design goals lead to the system architecture illustrated in Figure 1. Students and teachers interact with the system via their web-browsers.

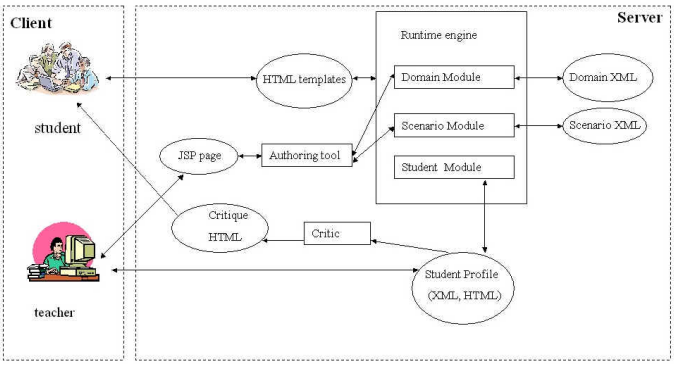

Figure 1: Indie system architecture.

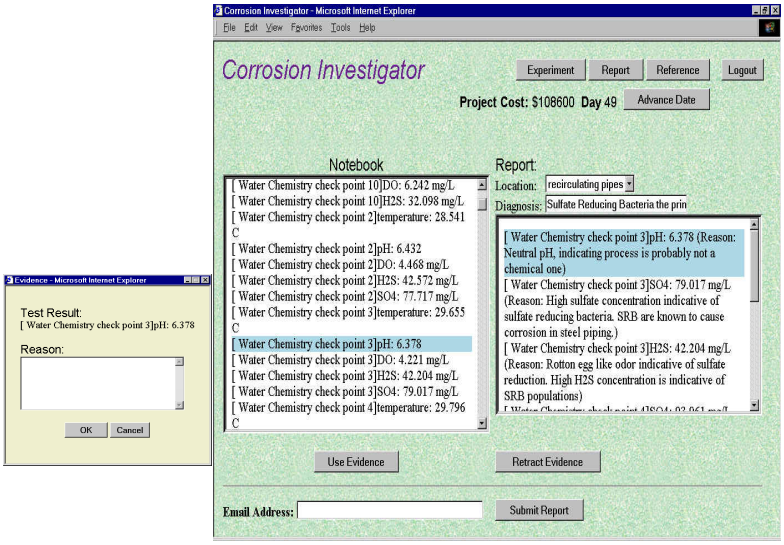

Figure 2: The Report screen in Corrosion Investigator.

Runtime Engine

The Java-based runtime engine is responsible for generating the environment where students perform the learning activity. The learning environment consists of a set of HTML templates, which include a welcoming template showing the "challenge" document; a Reference template where students can browse materials describing the scenario and domain content; an Experiment template where students can order tests and collect results; and a Report template (Fig. 2) where students can construct arguments for their diagnoses, using the test results saved in the Notebook (see Fig. 2) along with explanations. When students access any of these templates from their web-browsers, the runtime engine customizes the HTML template using JavaScript code generated on the fly according to the information in the domain module, scenario module and student module. For instance, the interface for students to specify parameters for a test will be generated based on the information about the test in the domain module. The content in the Notebook will be generated based on what test results students have received, which are contained in the student module. The runtime engine handles all the interactions between the system and students, e.g., automatically generating lab test results based on requests from students, facilitating the construction of arguments.

Domain Module

The domain module contains the information specified in the domain XML. It includes specification for domain-specific tests that students can run, and possible claims students will make. A test specification includes a link to the description page of the test, all the parameters for the test (such as Location of Sample and Primer Set for the DGGE test), available parameter values that a student can pick for each parameter (such as

Methanogens and Nitrifiers for the Primer Set parameter), and all the possible keywords that a student can use to retrieve the test. When a student input matches any keyword of a test, the test will be shown to the student. Students can then read the description of the test in the system and decide parameter values for the test. This approach gives no hint about what tests are necessary for solving the challenge, which leaves the activity totally open-ended. An alternative approach would be to have students select tests in a pull-down list. These tests would be realistic and common, but not all appropriate in the current scenario. Students would still need to make conscious decisions about which one to choose. At the request of the faculty member using Corrosion Investigator, we used the first approach. The effect of this part of the interface was evaluated and shown in the Evaluation and Results section.

Time and cost are two critical aspects for a real world project. To make the learning environment more authentic, we introduced them as test attributes. Tests may take a number of days to run, and students may need to wait several days (in simulated time) for the results to be available. Students have to plan the sequence of tests keeping time to a minimum. There are also costs assigned with each test. For example, each option for the "Primer Set" parameter of the FISH test costs five thousand dollars. Costs and time are two realistic mechanisms for getting students to think hard about when and what tests should be run. The amount of time and money a student spends in the project are recorded in the student module and made available for critiquing by the teacher.

Scenario Module

Detail information about how test results are generated and displayed under various test conditions specified by the students is contained in the scenario module. In order to make the learning situation authentic, the system allows test results to be generated randomly based on constraints specified in the scenario module. In this way, the same test may be run over and over again with the students getting different results; all of which are within parameterized acceptable range. Without changing the domain module, new scenarios can be created by modifying existing scenarios. Multiple scenarios can be created for the same domain, which improves the reusability of a completed domain module. Designing and implementing a domain module with a scenario module involves significant work that we consider only subject-matter experts and dedicated module builders can do. Extending or changing an existing scenario, however, takes the amount of work that a typical teacher might do in preparing for a new course, and with our authoring tool, additional programming is not necessary. The separation of the domain and scenario content makes customization of the learning environment feasible for teachers.

Student Module

The student module keeps track of student activities in the system. This includes tests that students have run (some have already gotten results back, some are in the queue waiting for results to be ready), claims and supporting evidence created, and the time and money that have been spent. Each student's profile is saved in XML format, which will be used to construct the student module during runtime. An HTML version of the file is created for teachers to critique students. Items, such as reasons for running a test and evidence points in the report, are clickable hyperlinks. When the teacher clicks on an item, a pop-up window allows the teacher to enter critique on the clicked item. All critiques are saved in a separate file (Critique HTML) for students to view. The Critic module handles all these interactions. While the software is not yet able to understand and provide feedback to input in natural language, having student activities in the learning environment accessible to the teacher in real-time fills in the gap and therefore complements the feedback generation from the computer.

Authoring Tool

The Indie learning environment provides both the interface and underlying code to enable students to do Investigate and Decide activities. Authors need only to specify the content of these activities, i.e., the challenge, tests, test results, and possible claims. This is similar to the approach used by Bell (1998).

Our system is written in Java. We used the JavaBean approach to generate the authoring interface. Tests, test results, etc., are defined as data objects. The interface is generated automatically from these object definitions (Fig. 3). The authoring tool first queries the object that the author selects to determine editable

properties of the object. We define properties that have both a set-method and a get-method as editable. For an editable primitive-type (e.g., int, double) and String-type property, a text field in HTML is generated. For a boolean type property, a menu with two options, true and false, is generate. For a property whose value is an object, a hyperlink with a servlet call telling the server which object the author wants to edit is generated. For any property whose value is a group of objects, the editing interface becomes a list showing all the values in the group. The author can add an item to the group, pick an item to edit, or delete an item. A JSP page places the dynamically generated HTML code inside a form. The author can do the editing in a web browser and submit the data to the server. The authoring tool maps the input onto the objects in the system and updates them accordingly using the JAVA reflection API.

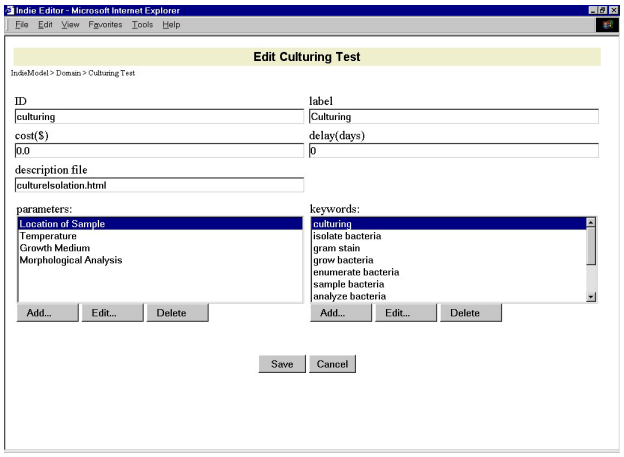

Figure 3: The authoring tool.

When the author wants to add a new item to the system, the system will provide a list of existing objects of the same type for the author to choose. Existing objects can serve as templates for creating new objects. Authors can make minor changes to an existing object and save it as a new object. Existing objects are cloned before being edited. Providing access to existing objects in the system helps the author to learn from previous examples and also saves work from starting from scratch.

The implementation of the authoring tool is independent of the content of the system. In other words, the authoring tool does not know any details of the content it operates on. Our approach is similar to the Naked Objects (Pawson & Mathews, 2002) approach in the sense that the authoring tool uses the JavaBean and Reflection technology to go through all the objects and provide an interface for each object when it is selected. Our approach differs in that only one object is presented at a time in the web-browser avoiding editing conflicts or inconsistencies.

The interface of Indie learning environments is made up of a set of HTML templates. The look-and-feel of the interface can be easily authored using an off the shelf webpage editor. These templates along with the authoring tool provide infrastructure and support for building important facilities for learning environments, such as persistent structured portfolios and argument construction tools. Compared to generic authoring tools, e.g. Director, WebCT and FrontPage, which assist authors mainly on the development of graphical interface, our authoring tool scaffolds the author in developing a resulting system that is based on sound educational principles. By using our tool, teachers are guided to create learn-by-doing project-based learning environments which we believe will lead to the improvement of the curriculum in educating future adaptive experts.

Evaluation and Results

In May of 2002, Corrosion Investigator was used in a class by six first-year graduate students in the Civil Engineering Department at Northwestern University. They were asked to form into two groups of three each, which resulted in a 3-male group and a 3-female group. The pipe corrosion challenge was introduced in class. After a week, each group gave a presentation on their thoughts about what might be causing the corrosion problem. Then, the Corrosion Investigator software was introduced to the students. After using Corrosion

Investigator for 3 weeks, students completed a survey on their experience.

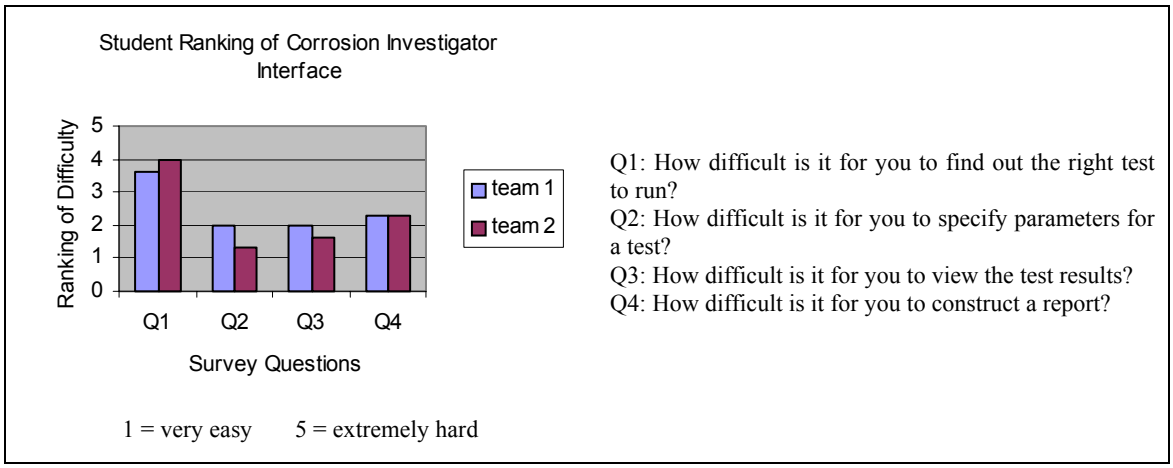

Figure 4: Student ranking of Corrosion Investigator interface.

Figure 4 shows student opinions of the Corrosion Investigator interface. The most difficult aspect of the interface to use was the mechanism for finding tests to run. No other parts of Corrosion Investigator were considered difficult. Improving the interface for finding tests to run becomes one of our future research issues.

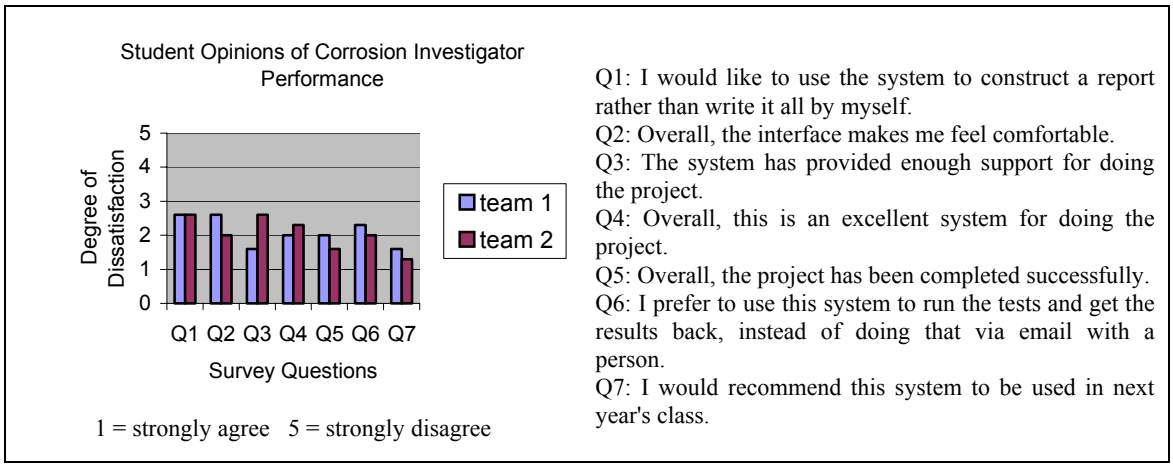

Figure 5: Student opinions of Corrosion Investigator performance

According to the students' responses shown in Figure 5, overall the system was satisfying for doing the project. According to the professor, use of the Corrosion Investigator significantly reduced his workload from 24 man-hours to 4 man-hours. This was most evident in reducing the work involved in generating the test results. At the same time, students also benefited from the immediate feedback generated by the system. Time savings from simulating the testing reduced the project time from 8 weeks (the time it took when all the test results were generated manually by the professor) to 3 weeks. According to the professor, the quality of the students' final reports using the Corrosion Investigator was identical in quality to those of the students when he taught the course without the software. Based on one criterion of educational effectiveness, which is the amount of time that students take to reach a certain level of achievement, Corrosion Investigator can be considered successful. Though the data was based on only six participants, it is still encouraging.

The authoring approach was proved to be successful in the previous version of Indie (Dobson, 1998). We hope to have the professor create a new scenario module for Corrosion Investigator using the authoring tool in May 2003.

Discussion and Future Work

Although the Indie learning environment employs a simulation to engage students in an authentic task (Brown et al., 1989), it is more than just an interactive computer-based simulation (ICBS) (Chou, 1998). An ICBS enables students to explore scientific phenomena in a learn-by-doing environment. If students, however, are not engaged in authentic tasks, the use of the knowledge and skills in real life remains unclear. An Indie learning environment addresses this issue by engaging students in learning activities where they need to investigate and develop artifacts and solutions to real problems. In order to accomplish some task, students learn the utility of domain knowledge in the context of a realistic challenge. In addition, previous scientific simulations did not provide effective ways for assessing students learning. The Indie learning environment requires students to provide explicit reasons for their moves in the system, e.g., running a test, using a test result as evidence for a claim. This allows the instructor to examine and critique the students' underlying understanding in an interactive environment.

Stand-alone intelligent tutoring systems (ITS) (Larkin & Chabay, 1992; Sleeman & Brown, 1982) have been in use for a long while. They offer individualized and just-in-time feedback. The feedback, however, is pre-defined and limited. The Indie learning environment includes teachers in the feedback loop, which leverages human expertise in providing high-quality feedback, while leaving room for computer-generated feedback.

We hope to have teachers use Indie to build learning environments in various domains. This development process will help us evaluate the flexibility of Indie, and at the same time, generate new learning-by-doing modules that can serve as examples in future deployment. During the entire developmental phases, the project is undergoing formative and summative evaluation. Students in the pilot classes will be given pre and post tests for their experience of the system. Faculty members who use Indie will evaluate the sufficiency, usability, appropriateness, and pedagogical quality of the tool.

Conclusion

In summary, we have described Indie, a software tool which includes an authoring tool and a content-independent engine for delivering web-based learn-by-doing environments where students run simulated experiments, analyze test results and construct arguments in an authentic setting. The architecture of the system allows teachers to access and critique student activities in the learning environment. It also provides an opportunity for teachers as non-programmers to customize the content of the learning environment. We presented preliminary results showing that Corrosion Investigator, an example of the Indie learning environment, proves to be beneficial for both the students and teacher in a class.

References

Bell, B. (1998). Investigate and decide learning environments: Specializing task models for authoring tools design. *Journal of the Learning Sciences*, Vol. 7. No. 1.

Bell, B. L., Bareiss, R., & Beckwith, R. (1994). Sickle cell counselor: A prototype goal-based scenario for instruction in a museum environment. *Journal of the Learning Sciences*, 3, 347-386.

Bransford, J. D., Brown, A. L., & Cocking, R.R. (Eds) (1999*). How people learn: Brain, Mind, Experience, and School.* Washington, DC. National Academy Press.

Bransford, J. D., Goldman, S. R., and Vye, N. J. (1991). Making a difference in people's ability to think: Reflections on a decade of work and some hopes for the future. In R. J. Sternberg and L. Okagaki (Eds.), *Influences on Children* (pp. 147-180). Hillsdale, NJ: Erlbaum.

Blumenfeld, P., Soloway, E., Marx, R., Krajcik, J., Guzdial, M., Palincsar, A. (1991) *Motivating Project- Based Learning: Sustaining the Doing, Supporting the Learning*, Educational Psychologist, Vol. 26, No. 3-4.

Brown, A. L. (1988). Motivation to learn and understand: On taking charge of one's own learning. *Cognition and Instruction*, 5, 311-321.

Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational Researcher*, 18, 32-41.

Chi, M. T. H., de Leeuw, N., Chiu, M., and LaVancher, C. (1994). Eliciting self-explanations improves understanding. *Cognitive Science*, 18, 439-477.

Chou, C. (1998) The effectiveness of using multimedia computer simulations coupled with social constructivist pedagogy in a college introductory physics classroom, *doctoral dissertation*, Teachers College-Columbia University, New York.

Cleary, C., & Schank, R., (1995). *Engines for Education.* Hillsdale, NJ: Lawrence Erlbaum.

Collins, A., Brown, J.S., & Newman., S. (1989) Cognitive Apprenticeship: Teaching the Craft of Reading, Writing, and Mathematics, In L.B. Resnick (Ed.) *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Dobson, W.D. (1998). Authoring Tools for Investigate and Decide Learning Environments*. Ph.D. thesis*.

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

Larkin, J. H., & Chabay, R. W. (Eds)(1992), *Computer-Assisted Instruction and Intelligent Tutoring Systems.* Hillsdale. N.J. Lawrence Erlbaum Assoc.

Palinscar, A. S., and Brown, A. (1984). Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and Instruction*, 1, 117-175.

Pawson, R., & Mathews, R. (2002) *Naked Objects.* Chichester, England: Wiley.

Qiu, L., Riesbeck, C. K. (2002) Open Goal-based Scenarios: an architecture for hybrid learning environments. *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education (E-Learn 2002)* Montreal, Canada, Oct. 15-19.

Schank, R. (1994). Goal-based scenarios: A radical look at education. *Journal of the Learning Sciences* 3, 4, 429-453.

Schank, R., & Korcuska, M. (1996) Eight goal-based scenario tools (Tech. Rep. No. 67). Evanston, IL: Northwestern University. The Institute for the Learning Sciences.

Schank, R., Fano, A., Bell, B., & Jona, M. (1993). The Design of Goal-Based Scenarios. *Journal of the Learning Sciences* 3:4. 305-345.

Sleeman, D., & Brown, J. S. (Eds)(1982). *Intelligent Tutoring Systems.* New York. Academic Press.

Acknowledgements

This work was supported primarily by the Engineering Research Centers Program of the National Science Foundation under Award Number EEC-9876363.