Semi-Automated Part-Task Trainer Prototype Development Environment

Rusty Kinnicut and Dick Stottler

Contract Number N68335-98-C-0147

Topic Number: N98-059
Topic Title: Interactive Part-Task Training over Local Area networks (LANS) and the Internet"

Technical POC: Naval Air Systems Command HQ
Attn: William Walker, PMA2052C
Bldg. 2272 - Suite 345
47123 Buse Road Unit IPT
Patuxent River, MD 20670-1547

Final Progress Report

January 20, 1999

Stottler Henke Associates, Inc. (SHAI)
1660 So. Amphlett Blvd., Suite 350
San Mateo, CA 94402

Semi-Automated Part-Task Trainer Prototype Development Environment

Abstract

In this Phase I SBIR research, we proved the feasibility of a semi-automated part-task trainer (PTT) prototype development environment. In this effort, we set out to address the shortfalls of traditional PTTs and to introduce new concepts into the PTT prototyping process. The research developed innovative rapid prototyping techniques for the development of PTT prototypes. These techniques are leveraged upon the state-of-the-art software engineering methodologies of component oriented programming (COP), distributed objects, and visual development environments. This means our environment will maximize software reuse and support a number of COTS tools. This innovation will result in a dramatically reduced cost of PTT prototype development and maintenance. Further, prototypes built with this system will employ intelligent tutoring systems (ITSs) which tailor training programs based on detailed mental models of the students. Techniques for defining ITSs using the COP paradigm were designed during this project. Finally, the prototyping environment enables the development of PTTs which take full advantage of intranets and the Internet. This enables features such as distributed simulations, team training, and student monitoring utilities to be integrated into PTT prototypes. A proof-of-concept prototype was developed to demonstrate the key concepts and a full operational system was designed.

Semi-Automated Part-Task Trainer Prototype Development Environment

Final Report

1. INTRODUCTION **2**

1.1 PHASE I OBJECTIVES 3

2. PHASE I INVESTIGATION **4**

2.1 CURRENT PART-TASK TRAINERS 4
2.2 RAPID PROTOTYPING TECHNOLOGY SURVEY 6
2.3 INTELLIGENT TUTORING SYSTEMS AND PART-TASK TRAINING 8
2.4 PROOF-OF-CONCEPT PROTOTYPE 9
2.5 PHASE II SYSTEM DESIGN 10

3. PHASE I PROTOTYPE **11**

3.1 NAVIGATIONAL PROCEDURES OF AH-1W CDNU 11
3.2 PROOF-OF-CONCEPT RAPID PROTOTYPING 11
3.3 DISTRIBUTED ARCHITECTURE 13
3.4 INTELLIGENT TUTORING SYSTEM 14
3.5 SIMULATION ENVIRONMENT 17
3.6 INSTRUCTOR UTILITY 18

4. PHASE II SYSTEM DESIGN **18**

4.1 OPERATIONAL TRAINER-PDE PROTOTYPE 18
4.2 TRAINER-PDE SYSTEM OVERVIEW 19
4.2.1 Components *20*
4.2.2 Palettes *20*
4.2.3 Event Model *21*
4.2.4 Property Editors *21*
4.2.5 ITS Architecture Template and Development Roadmaps *22*
4.3 PHASE II SYSTEM ARCHITECTURE 22
4.3.1 Platform Specification *22*
4.3.2 Development Environment *22*
4.3.3 Component Interaction Diagram *23*
4.4 TRAINER-PDE SOFTWARE COMPONENTS 24
4.5 PHASE II SYSTEM DESIGN SUMMARY 27

5. FUTURE WORK & COMMERCIALIZATION **27**

5.1 PHASE II TECHNICAL OBJECTIVES 27
5.1.1 Improve Efficiency of PTT Development and Effectiveness of Resulting PTT *28*
5.2 COMMERCIALIZATION PLANS 29

1. Introduction

The need of the armed forces to maintain effective readiness at a low cost is as necessary as ever. Part-task training plays a crucial role in maintaining this readiness. The more resources these part-task trainers can make use of and the more realistic the simulation environment they provide, the more effective the part-task trainer will be. The ideal part-task trainer will make efficient use of state-of-the-art concepts to attain these goals. This is where current part-task trainers typically fall short. Most part-task trainers are built in a machine-dependent manner for platforms which can't effectively fulfill the demand placed on them by the part-task trainer. This machine dependence deprives the part-task trainers of forward compatibility and portability. Forward compatibility and portability are properties which would enable part-task trainers to gracefully evolve with technological advancements. Machine dependence also insures expensive part-task trainer maintenance cost when modifications are made to a training program. That is, when the task being trained is even slightly modified, the cost of supporting the changes within the part-task trainer is very high. Further, current part-task trainers fail to take efficient advantage of the networked computing environments which prevail today. When current part-task trainers make use of computer networks, they typically ignore security issues. The final shortfall of traditional part-task trainers is their failure to employ some of the powerful concepts developed by computer-based training researchers in recent years.

What is needed is a robust part-task trainer prototyping environment which effectively addresses all of the above concerns. If this environment makes proficient use of an industry proven component model, a standard distributed object model, and a robust portable programming language, then the issues stemming from machine dependence will have been effectively addressed. The environment can be visually driven and distributed over a local area network (LAN) or a wide area network (WAN) securely. The environment will be sufficiently general and extensible, thus supporting the easy integration of many value-added enhancements. Finally, the integration of a number of computer-based training methods will enable the resultant part-task trainers to tackle new classes of training tasks not previously considered for part-task training.

The Phase I research set out to address the concerns with current part-task trainers and to develop a new concept for part-task trainer prototyping. The new concept for part-task trainer development is the Semi-Automated Part-Task Trainer Prototyping Environment, Trainer-PDE. Trainer-PDE leverages the use of emerging, industry proven technologies, from the object oriented community to facilitate the rapid prototyping of part-task trainers. These industry proven technologies allow for Trainer-PDE to make efficient use of relatively low-cost Commercial-Off-The-Shelf (COTS) software, reusable software components, and open standards. Additionally, support for Intelligent Tutoring Systems (ITS) which cater training programs to individual students, plays a significant role in the Trainer-PDE concept. Intelligent tutoring systems enable the part-task trainer to tackle more cognitive training tasks than the traditional part-task trainer. Intelligent tutoring systems can also often make up for shortfalls in the realism of low fidelity simulation environments. The Trainer-PDE system will result in the ability to efficiently build high quality part-task trainers which take advantage of modern computer networks and overcome the limitations of traditional Part-Task Trainers. The effective use of emerging technologies addresses the development issues of part-task trainer prototyping and the use of intelligent tutoring systems enables part-task trainers to tackle new classes of training. In summary, an end Trainer-PDE system will enable the development of highly effective part-task trainers at a low cost.

The next sub-section (1.1) outlines the technical objectives of the Phase I research; section 2 discusses the Phase I research activities; section 3 presents the proof-of-concept prototype; section 4 presents the Phase II system design; and section 5 describes the future of

this project, including Phase II objectives and eventual Phase III commercialization. The appendix provides screen shots and examples from the Phase I proof-of-concept prototype.

1.1 Phase I Objectives

There were six major technical objectives of the Phase I effort. Each of these Phase I goals were accomplished.

A. Automate part-task trainer prototype design and development.

Our primary task for the Phase I was to develop a concept to automate or semi-automate part-task trainer prototype design and development. The hypothesis is that by taking advantage of the similarities between various part-task trainers, portions of the part-task trainer prototyping process will be automated. The other objectives for this Phase I were essentially sub-objectives towards the achievement of this goal. A comprehensive understanding of current training environments was required to determine similarities and differences. Identification of key emergent technologies was required to determine how exactly these similarities and differences could be exploited. The feasibility study of various computer-based training (CBT) paradigms was necessary to push the envelope of tasks applicable to part-task training. This study also examined the hypothesis that the effective use of CBT paradigms will allow effective part-task trainers to be built on less expensive, low fidelity hardware (e.g., a ruggadized laptop). A proof-of-concept prototype was required to prove the feasibility of Trainer-PDE and to clearly demonstrate the concepts involved. Finally, these objectives are combined and put forth in the formal system design of a Phase II Trainer-PDE operational prototype.

B. Obtain a comprehensive understanding of part-task trainer prototype development.

This objective was the priority early in the Phase I research. The success of the other Phase I objectives were highly dependent on obtaining a comprehensive understanding of part-task trainers and part-task trainer prototype development. This objective involved identifying the types of tasks currently trained using part-task trainers. The objective also included a study into the shortfalls of these traditional part-task trainers. This study included both shortfalls as far as the technological concepts traditional part-task trainers employ, as well as the shortfalls in the traditional part-task trainers effectiveness as training tools. After these shortfalls were identified, objectives C and D sought to address them.

C. Identify the appropriate state-of-the-art products and techniques which will fulfill the security, distribution, portability, and extensibility requirements.

This objective started early in the Phase I and continued throughout the research effort. The success of this objective was necessary to prove the feasibility of a rapid prototyping environment for part-task trainer development as well as the benefits of distributed part-task trainers over a LAN or WAN - two core concepts of the Trainer-PDE system. Early on we recognized how recent achievements in software reuse, open standards, and visual development environments were resulting in software engineering tools which blurred the lines between the traditional software development cycles. By successfully designing a system for part-task trainer prototyping which emphasized these state-of-the-art software engineering technologies, we will effectively produce a paradigm for the *rapid* prototyping of part-task trainers. We also recognized how this new development methodology will effectively address the majority of concerns which plague traditional part-task trainer development and maintenance. That is, the same techniques which enable rapid prototyping also increase the portability and maintenance of the end part-task trainer prototypes.

D. Prove the feasibility of rule-based, scenario-based, and simulation-based computer training paradigms.

This objective was pursued after the thorough study of current part-task trainers was completed. The primary goal was to integrate the concepts of intelligent tutoring systems and student modeling with part-task training. Stottler Henke Associates, Inc. has a significant amount of experience applying intelligent tutoring systems to real world problems. Our aim was to use this practical experience to apply ITSs to the domain of part-task training. The success of an intelligent tutoring system depends heavily upon the ability to build an accurate mental model of the student and then catering the training program based on this mental model. Techniques were developed for describing these mental models within a visual environment. Mechanisms were developed to enable the ITS to monitor a student's performance within a simulation, gauge the student's understanding of principles, and specialize the training program based on these factors. These techniques were designed to work with the rapid prototyping methodologies identified by objective B.

E. Develop a proof-of-concept prototype.

In order to test our theories, and as a means of demonstrating the success of our work, a proof-of-concept prototype was developed. The prototype touched on each of the principles relevant to the success of this project and served as a mechanism for relaying these concepts to others. The prototype proved invaluable when demonstrated to some personnel of the HMT-303 helicopter training school at Camp Pendleton. The feedback we received verified the direction of our project and re-affirmed the importance of our chosen objectives. Additionally, the development of the prototype served as a testbed for experimenting with the various software engineering technologies and training concepts involved. This experimentation proved very useful when designing the Phase II system architecture. A detailed overview of the proof-of-concept prototype is provided in section 3.

F. Design architecture of full-scale, semi-automated, part-task trainer prototype authoring environment.

The successful completion of the other objectives was a requirement for the design of the Phase II system. This objective was the final objective completed in the Phase I effort. A complete system design incorporating state-of-the-art rapid prototyping concepts and the integration of intelligent tutoring systems with part-task training was completed. A detailed overview of the system design is presented in section 4.

2. Phase I Investigation

In order to accomplish the technical objectives discussed in section one, Stottler Henke Associates, Inc. embarked upon a series of research activities along the various related tracks. This section outlines this experience and presents the results of this investigation. The following two sections (sections 3 and 4) will describe in detail the two major products of the Phase I research, the Phase I proof-of-concept prototype and the Phase II system design.

2.1 Current Part-Task Trainers

During Phase I, we investigated the current state of part-task trainers and part-task trainer development. We struggled for some time trying to pinpoint how exactly part-task trainers were currently developed and for which specific tasks they were used. We knew that an important requirement of this project is that the part-task trainers built with Trainer-PDE operate within highly portable, ruggadized laptops as well as desktop machines. This requirement imposed certain restrictions on the fidelity of the simulation environments we could develop for our part-task trainers. From the initial investigation of part-task trainers, which involved a

literature search as well as phone interviews with part-task trainer developers, we learned that the term *part-task trainer* is used for a variety of vastly different training tools. These training tools included nearly everything from full blown 6 degrees of motion simulators to small-scale models of devices to virtual mockups of instrument panels on computer screens. This variety initially made it difficult for us to pinpoint our goals. However, with subsequent trips to the Naval Air Station in Patuxent River and then to the helicopter training school in Camp Pendleton, we were able to gain a firm grasp of the target domain and identify a real world need for a Trainer-PDE system. This accomplishment was a major milestone in the Phase I research. Further, the connections we established during this research track offer a tremendous benefit to future research by generating potential Trainer-PDE prototype users and access to instructors.

The COTR guided us towards the H-1 training community for evaluating current part-task trainers and for identifying potential target users of an end Trainer-PDE system. The pilot seat of the AH-1W Cobra attack helicopter is perhaps one of the busiest seats in the armed forces. It is estimated that the pilot of the AH-1W must handle 1.7 times the equipment of a typical Navy pilot. The cockpit is filled with a myriad of loosely coupled navigational equipment, communication devices, sensors, and weapon systems. The pilot must maintain knowledge of all of these devices while piloting the helicopter. Most all of these individual devices are suitable target domains for part-task trainers. However, because traditional part-task trainers are expensive, there are very few part-task trainers for these devices available. Further, the flight students and instructors have reported serious flaws in the part-task trainers currently available. They have all but dismissed their current part-task trainers as useless. During this investigation, we worked with several instructors and identified several flaws in current part-task trainers and identified the properties which would make these part-task trainers effective. We believe Trainer-PDE part-task trainers will successfully address these flaws. Most of this work took place at the helicopter training school, HMT-303, in Camp Pendleton.

A principle problem found with current part-task trainers is that there is very little, if any, feedback offered to the trainee. This makes the part-task trainers very uninteresting to use. As a result, many of the part-task trainers we looked at are not used very often, if they are used at all. The instructors and students we spoke with recognized that if some sort of engaging multimedia feedback were given to the students, then the part-task trainers would be infinitely more useful. A second major flaw is related to the fact that these part-task trainers were developed in a highly machine dependent manner. Occasionally, the software of the actual devices within the helicopters change. When this happens, the part-task trainers model of the device is not usually updated to reflect these changes. This means that the part-task trainers do not behave the same as the devices they are designed to train. It is clear how this can be problematic.

After the more general survey of the part-task trainers at HMT-303, we moved forward and focused in on the navigational functions of the Control Display Navigation Unit (CDNU) of the AH-1W. This particular device was chosen because, though HMT-303 already had part-task trainers for this device, they are not very useful and are barely used by the students. This provided an opportunity for us to demonstrate how a useful part-task trainer can be built for this device. The part-task trainer must overcome the limitations of the current CDNU part-task trainer. Namely, it must engage the user and it must be easy to modify such that it can stay up to date with changes made to flight software. Further challenges included demonstrating the benefits of delivering part-task trainers over a computer network as well as demonstrating the benefits of the use of student modeling and intelligent tutoring systems within part-task trainers. A second reason this device was chosen to focus our research was because we felt that by focusing only on certain navigational procedures it could be scaled nicely to fit a two-month proof-of-concept prototype development effort. Also, we recognized that the knowledge learned

from this phase I effort could be directly applied to the Phase II development of an operational prototype which covered all navigation and communication procedures of the CDNU.

After performing this evaluation of current part-task trainers and part-task trainer prototype development, we moved forward to determine exactly how emerging technologies for rapid prototyping and intelligent tutoring systems could effectively address the problems of traditional part-task trainers. The feasibility of the developed methods were then demonstrated in a proof-of-concept prototype and documented in the design of an operational rapid prototyping environment for part-task trainer development.

2.2 Rapid Prototyping Technology Survey

During Phase I, we performed a survey of emerging technologies for rapid prototyping and, more specifically, rapid prototyping for part-task trainers distributed over computer networks. A general investigation of various emergent technologies was performed early in the project. This initial survey was kept general because we were still gathering knowledge on current part-task trainers to enable us to focus our research. After more knowledge was gained on part-task trainers, and especially after the selection of the CDNU as a suitable domain for part-task training, we began to focus our research. The types of relevant technologies examined included Computer Aided Software Engineering (CASE) tools, Component Oriented Programming (COP) methodologies and environments, distributed object model methodologies and implementations, portable programming languages, secure communications protocols, and various application programming interfaces (API) for building interactive simulations. In addition to this investigation of development environments and utilities, we also examined the literature describing these technologies as well as rapid applications development (RAD) in general. The objective of this survey was to identify how these environments could be used to develop a rapid prototyping environment capable of overcoming the shortfalls of current part-task trainer prototyping.

The hypothesis which initiated this research is that these new technological advancements in software development are blurring the lines between traditional development cycles. Object oriented analysis and design tools such as Rationale Rose and Advance Software GDPro currently employ the formal object modeling language, UML (Universal Modeling Language). These object modeling tools can be used to generate source code from UML object models or generate UML object models from source code. This type of reverse engineering resembles a step towards bridging the gap between the design and implementation phases of software development.

Component oriented programming is a refinement of object oriented programming, which defines an open standard interface for software objects, or components. Because this interface conforms to an open standard, the methods and properties of the objects (or components) can be realized at runtime. The importance of this is that it enables instances of the components to be manipulated and defined at runtime. The capability to manipulate instances of components at runtime enables components to be used within a visual development environment to build applications. Further, it greatly enhances the reusability of the software components. Software reusability is debatably one of the false promises of the early object oriented movement. COP makes reusability a reality. The ability to visually manipulate components within a COTS tool in order to build software applications is another step towards bridging the gap between the architectural design and implementation Phase of software development. The property that components can be manipulated as runtime is a step towards bridging the gap between the implementation and testing phases of software development.

However, during our survey we failed to find a CASE analysis and design tool capable of generating and reverse engineering code which conformed to a standard component model.

The development of such a tool would enable a link between the analysis, functional design, architectural design, implementation, and testing phases of software engineering. However, as all of these standards and utilities are still maturing, such leaps are not yet possible. A utility to bridge these stages was considered for development, however, it was decided that this would be outside of the scope of this project. Because the technology was more mature and more directly related to part-task trainer prototyping, we made the decision to focus on component oriented programming and COTS visual development environments. An additional reason to focus on these technologies is because they are closely related to another area of interest of this Phase I - distributed object models.

Component oriented programming has received a huge amount of attention over the past year. The introduction of the JavaBeans component model is agreed by many to be the most significant event in the language's history and is expected to be the catalyst that moves Java usefulness beyond simple web-based applications. We have already seen this, to some degree. Many serious Java applications have been developed over the past year. The number of commercial-off-the-shelf software which support the development and use of the JavaBeans components is further evidence of industry interest. At the time this investigation was performed, many of these tools were still in their early stages of maturity. IBM's Visual Age for Java version 2.0 was selected for the implementation of the proof-of-concept prototype for several reasons. These reasons include the fact that the JavaBeans component model is relatively easy to learn, IBM's tool was determined to be more stable than many others evaluated, and Java as a programming language works well within a networked environment.

Though the JavaBeans component model was chosen for the Phase I research, there are several other major component models in use today. A re-evaluation of JavaBeans and other component models should be performed during the Phase II. Each component model has different advantages and disadvantages when compared to one another. The objective of the Phase I survey was to determine the feasibility of using component oriented programming for the rapid prototyping of part-task trainers, not to determine which component model was the *best*. In short, JavaBeans has the advantage of being widely supported, mostly platform independent, and possess the ability to work very well in a networked environment. Issues regarding the efficiency of Java are being addressed by just-in-time (JIT) compilers; however, there are still some concerns. Microsoft's distributed component object model (DCOM) has the advantage of being highly efficient and easy to use when used within other Microsoft products and for the development of applications for the Microsoft operating systems. The negative side includes a high learning curve for building new components and a serious lack of portability. The Common Object Request Broker Architecture (CORBA) effectively defines a component model of its own. CORBA is both language and system-independent and boasts a highly robust distributed object model. However, to date, CORBA as a component oriented programming model is not widely supported. CORBA also suffers from being less efficient than DCOM and a fairly significant learning curve. COTS Tools to address these issues are expected to emerge over the next couple of years.

Another major technology surveyed included distributed object models. The three major distributed object models examined include CORBA, DCOM, and Java Remote Method Invocation (JavaRMI). The pros and cons of these distributed object models are roughly the same as the corresponding component models. One difference, however, is that CORBA as a distributed object architecture is probably the most robust and widely supported of the three. JavaRMI has the advantage of an easy learning curve, but disadvantage of efficiency. DCOM has the advantage of efficiency, but disadvantage of platform dependence. It is important to recognize that all three of these distributed object models can work together quite well through the use of *bridges*. Bridges are tools which link one distributed or component object model to another. This means we don't necessarily have to tie ourselves to one distributed object model.

For the Phase I prototype, JavaRMI was chosen to demonstrate the advantages of a distributed part-task trainer over a LAN or WAN. This choice was made primarily because JavaRMI is probably the easiest methodology to use and this made it appropriate for use in a two-month development effort. For a Phase II operational prototype, however, the robustness of CORBA may be desirable. CORBA and the JavaBeans object model work very well together. In fact, it is anticipated that CORBA support will be adopted into the Java language specification in the near future. As such, the use of JavaBeans and CORBA may be wise implementation choices for the Phase II. Nonetheless, this issue will be revisited in the design refinement stages of a Phase II effort.

2.3 Intelligent Tutoring Systems and Part-Task Training

During Phase I, we developed techniques for enhancing the effectiveness of part-task trainers, through the use of an intelligent tutoring system. Intelligent Tutoring Systems track the mental models of students and specialize the training program based on this knowledge. The general flow of information within an intelligent tutoring system is depicted in Figure 1.

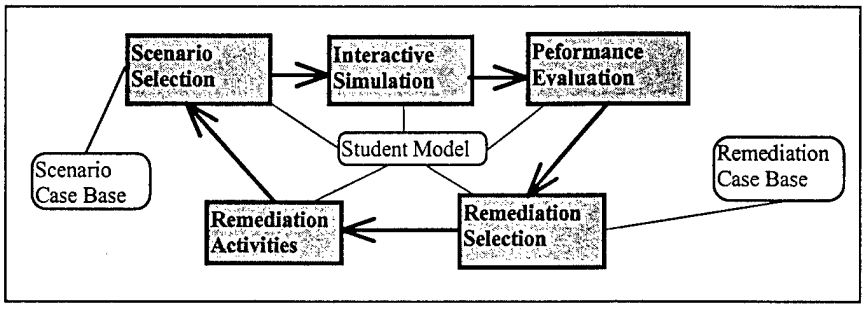


Figure 1. Information Flow in an Intelligent Tutoring System

The data structures shown in the boxes with rounded corners include a *scenario case-base*, *student model*, and *remediation case-base*. The student model maintains a hierarchy of principles along with a rating of the students understanding of each principle. The scenario case-base maintains a database of scenarios indexed by the principles they help present. The remediation case-base maintains a database of remediation activities also indexed by the principles they present. The *scenario selection* module uses the student model and case-based reasoning to select an appropriate scenario from the scenario case-base. The student is then presented the scenario within an interactive simulation. After the student performs the scenario within the simulation, the student's performance is evaluated and his or her student model is updated based on the principles achieved and missed by the student within the scenario. The *remediation selection* module then uses the updated student model to select the appropriate remediation activities from the remediation case base. The student then performs these activities and his or her student model is once again updated to reflect this activity. If there are still unlearned principles, the student is brought back to the scenario selection module and enters the cycle again.

The ITS cycle described has been successfully applied by SHAI to many different types of training tasks. The challenge of this portion of the Phase I research was to apply it to part-task training and, more importantly, to enable the creation of ITSs for part-task training within a rapid prototyping environment. Typically, when we implement an intelligent tutoring system, a large portion of the time is spent on developing a *principle hierarchy* for the domain. A principle hierarchy is a hierarchical breakdown of the various principles being trained and their relationships with one another. This principle hierarchy is extremely important because it determines how the students' mental models are to be built and maintained by the ITS. For this project, however, the emphasis is on building an authoring tool to facilitate the development of

intelligent tutoring systems and part-task trainers. So for our research, we developed a very simple principle hierarchy of the navigational principles of the CDNU. This simple principle hierarchy provided us with a means of testing different ideas for ITS authoring within the context of a rapid part-task trainer prototyping environment. Additionally, a simple scenario case base, sample scenarios and remediation modules were developed. These will be described in more detail in section 3 of this document, which describes the Phase I prototype.

Using this as a starting point, we were able to develop methods for defining the sample data within the part-task trainer prototyping environment. The techniques developed are general enough that they are applicable to virtually any part-task training domain. One or more reusable software components were designed for each of the modules shown in Figure 1. Each of these software components can be specialized for different part-task training domains through the use of custom *visual* editors. An additional benefit of the approach developed is that it is relatively easy to re-use the same components used for building the ITS for building utility applications geared towards the instructors. These utility applications allow instructors with no programming experience to develop and maintain principle hierarchies, the scenario case base, and the remediation case base.

A demonstration of these techniques is provided with the presentation of the Phase I prototype, in section 3. The techniques are formally described with the presentation of the Phase II system design, in section 4.

2.4 Proof-of-Concept Prototype

During the Phase I, we implemented a proof of concept prototype which demonstrates the feasibility of a full-scale Trainer-PDE system. This Phase I prototype is presented in detail in section 3. The principle objectives of the Phase I prototype is to validate the part-task trainer prototyping techniques developed and to demonstrate the feasibility of a complete Phase II system. In order to accomplish these goals, we designed the prototype such that it touched on each of the major issues involved with this project. This included the use of reusable software components for rapid part-task trainer prototyping, the use of a distributed object model for distributing the part-task trainer across a LAN or WAN, and the integration of intelligent tutoring systems with part-task training. The construction of the proof-of-concept prototype enabled us to obtain two secondary objectives. The proof-of-concept prototype would serve as a suitable testbed for experimenting with the various concepts involved in this project. This testbed was particularly useful when experimenting with different ways to integrate an ITS with a part-task trainer prototype. Additionally, the proof-of-concept prototype served as a valuable tool for relaying the ideas developed within this project to potential commercial users. An early version of the prototype was demonstrated to personnel in the computer-based training department of HMT-303 at Camp Pendleton. In addition to receiving valuable input, we also received endorsements and potential users for a Phase II effort.

The design and development of the prototype was almost entirely dependent on the completion of other major Phase I research tasks. These tasks included obtaining a comprehensive understanding of part-task trainers, completing a survey of rapid prototyping technologies, and developing techniques for defining an ITS within a visual development environment. After these tasks were completed, or nearly completed, we spent several months fine tuning the techniques and proving their feasibility through the Phase I prototype. The Phase I prototype developed JavaBean components for use within IBM VisualAge for Java version 2.0 to build a part-task trainer prototype for the navigational procedures of the AH-1W CDNU. JavaRMI was used to demonstrate the benefits of a distributed part-task trainer over a LAN or WAN. Also demonstrated, was how the use of a distributed object model and component oriented programming techniques could abstract away many of the complexities of network

programming. In designing the Phase I prototype, we took into consideration the task of designing the complete Phase II system which would occur during the last month of Phase I research. We did this such that our Phase I design would be scaleable to a Phase II system design, and so the phase I prototype not only demonstrates the feasibility of the end Phase II system, but it goes further and validates the Phase II system design. The prototype was completed a month before the end of the project completion date.

The navigational procedures of the AH-1W CDNU was selected as a target for the proof-of-concept prototype for several reasons. First off, the instructors and personnel at HMT-303 had identified several flaws with their current CDNU part-task trainers. We felt that successfully addressing these flaws would be a strong way to prove the feasibility of our approach. Other factors included the fact that the domain of the CDNU is highly scaleable. This feature enabled us to scale the domain to a size reasonable for a two-month development effort. Additionally, we found a lot of information about the CDNU within AH-1W NATOPS document. This important consideration enabled us to build a fairly accurate model of some of the CDNU functions. Finally, we felt that a CDNU part-task trainer could touch on each of the technological issues we were interested in demonstrating in this project.

The prototype developed involved a set of JavaBeans components which could be linked together within a COTS environment to build a CDNU part-task trainer. The part-task trainer built included an interactive simulation which could be distributed to several simultaneous users over a network. The part-task trainer also included an intelligent tutoring system which selected scenarios, monitored student performance, and maintained a student model database. Sample multimedia remediation screens were also demonstrated. In many cases, custom user interfaces were developed for the components used to build the part-task trainer prototype. Finally, a sample instructor utility was developed to demonstrate how an instructor might interact with students over a network of part-task trainers.

2.5 Phase II System Design

During Phase I, we developed a system design for the Phase II Trainer-PDE system. This task essentially started when we began the design of the Phase I proof-of-concept prototype. After completing the implementation of the proof-of-concept prototype, we evolved the Phase I design into the Phase II System design presented in section 4. Though the task of designing the Phase II system grew out of the phase I proof-of-concept prototype design, the task was not entirely straight-forward. The proof-of-concept prototype included a proof-of-concept Trainer-PDE system as well as a proof-of-concept CDNU part-task trainer. The design of the Phase II system, however, includes the design of the Trainer-PDE system as well as descriptions as to how to use it to build part task trainers, in a more general sense. To do this, we had to define a general framework capable of supporting the majority of part-task trainer prototypes and then describe how the prototyper should go about using the framework and Trainer-PDE to implement the part-task trainer. The majority of the less reusable software components will be in the interactive simulation portions of the part-task trainer prototypes. We also describe how reusable software components can be designed, implemented, and used by new interactive simulations. By doing this, the prototype developer will accumulate a library of reusable components which will aid in the design and development of future part-task trainers. More importantly, the use of reusable components will considerably facilitate the maintenance of the part-task trainers developed with Trainer-PDE.

Some principle objectives of creating a full system design of the Phase II system include providing an effective means of documenting our Phase I work and to give us a head start in a Phase II development effort. The Phase II system design is presented in section 4.

3. Phase I Prototype

This section provides an overview of the proof-of-concept prototype developed during the Phase I. The sub-sections refer to screen shots in Appendix A.

3.1 Navigational Procedures of AH-1W CDNU

The target domain of the proof-of-concept prototype is the navigational procedures of the AH-1W CDNU. The CDNU is a device on the AH-1W helicopter which is used for a variety of navigation and communication tasks. The CDNU on the AH-1W is comprised of an alpha-numeric keypad along with function keys and a small text display area. The CDNU interfaces a number of navigational and communications equipment including sensors, radios, radar, TACAN, and more. The pilot navigates through the CDNU data pages and uses the keypad to enter information. The CDNU then uses this information to configure the appropriate sensors or radios. The CDNU is also available to display various information about the equipment it interfaces. For example, a typical task the CDNU is used for, is entering a longitude/latitude location as a destination along with a desired time of arrival. The CDNU will then compute the direction and speed the pilot must fly his or her helicopter in order to arrive at that location at that time.

To limit the scope of our work such that it could be completed in a two-month development effort, we chose to focus on a limited set of the navigation procedures. *Table 1* lists the navigation procedures we chose to cover in the Phase I proof-of-concept prototype.

Table 1. CDNU Concepts within Proof-of-Concept Prototype

• **Waypoint**	• **Exp. Square Pattern**
• **Target Point**	• **Flight Plans**
• **Routes**	• **Update Progress**
• **PIMs**	• **Set Time**
• **Ladder Pattern**	• **Set Date**
• **Sector Pattern**	

One of the reasons why the tasks in *Table 1* were selected for this prototype was because they were believed to be fairly easy to model within a low fidelity simulation. Also, though these tasks are fairly basic and straight-forward concepts, they represent some of the more common navigational tasks the CDNU is used for. Therefore, it is very important that a pilot can routinely perform these sorts of tasks. A principle goal of the part-task trainer is to internalize these procedures into the AH-1W pilot to the extent that he or she can perform them with very little conscious effort. This will free up valuable cognitive resources for more of the pressing tasks of piloting the helicopter and/or assessing the current tactical situation.

To learn all the necessary information regarding the AH-1W CDNU, we primarily referenced the AH-1W NATOPS manual. When questions arose, we would direct them to contacts at HMT-303. This domain for part-task training served us well for the proof-of-concept prototype. The domain would also be a strong candidate for an operational prototype Phase II system. A phase II system would train the full suite of CDNU tasks including all navigation and communication procedures.

3.2 Proof-of-Concept Rapid Prototyping

In order for the Phase I to be successful, it was important to demonstrate how rapid prototyping techniques could be used for part-task trainer development. In order to demonstrate

this, we developed a proof-of-concept prototyping environment for part-task trainer development. This proof-of-concept prototyping environment consisted of all the necessary components to build the scaled CDNU part-task trainer. Essentially, the rapid prototyping environment was a very scaled down version of what the end Phase II system will look like. The prototyping environment developed only needed to focus on the components required to build a CDNU part-task trainer, as opposed to a Phase II system which must be general enough to build essentially any part-task trainer or, at the very least, provide a significant head start on the prototyping process.

The Phase I rapid prototyping environment consisted of a set of custom JavaBean components and a COTS tool which supported the JavaBean object model. The Java Development Kit version 1.1.6 was used to compile the custom JavaBeans. The components were organized into palettes and linked to build part-task trainers within IBM VisualAge for Java. Theoretically, any of the many COTS tools which supported the JavaBean component model could have been used for this purpose. Some of these other COTS tools include Inprise Jbuilder 2.0, Sun Java WorkShop 2.0, Lotus BeanMachine, Symantic Corp. Visual Cafe for Java, NetBeans 2.0, and many more.

Within the protoyping environment each component is associated with a set of properties along with property values, custom property editors, events accepted, and events fired. The prototype developer selects the desired component from a palette and places it on the canvas. The prototype developer then specializes the component by assigning values to the components properties. The task of assigning values to the component instances properties is facilitated by custom, user friendly, property editors. Communication between the various components is specified through the event model. Each component specifies the types of events it fires and the types of events it knows how to handle. The user then uses the COTS tool to graphically draw connections between components and specify which events traverse these connections.

For example, we developed a platform component to represent the state of a platform, such as an AH-1W helicopter. We then created an instance of this platform component to represent the helicopter which the CDNU is located in. We used a property editor to set location, heading and speed information, amongst other properties for the platform component. The platform component type fires a platform event every time one of these properties change. A CDNU component and map component were designed to know how to handle these platform events. The CDNU component uses the platform event information to update its internal data and to display information about the platform. The map component uses the platform event information to redraw the map and illustrate the new location of the platform. To enable these components to interact as desired, we next drew connections between the instance of the platform component and the instances of the CDNU and the map components. This connection specifies that platform events should be passed from the platform component instance to the CDNU and map component instances. Suppose now that we wish to instantiate another platform component to represent the aircraft carrier platform the AH-1W should land on. We draw a connection from this new platform instance to the map display component. The map component will now display the location of the aircraft carrier as well as the helicopter. Note, however, that we do not draw a connection from this new platform instance to the CDNU component, because the CDNU has nothing to do with this platform. This example is illustrated in Figure 2.

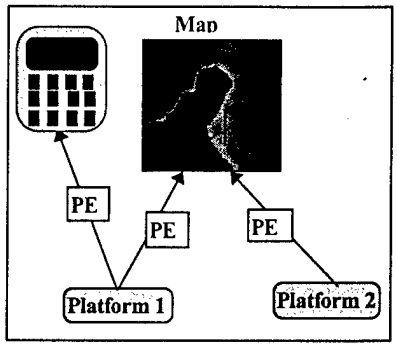


Figure 2. Event Model Example

The components for building the AH-1W part-task trainer were divided into palettes, based on their functionality. These palettes included *multimedia* (TPDE-MM), *simulation* (TPDE-SIM), *familiarization* (TPDE-FAM), *intelligent tutoring system* (TPDE-ITS), and *networking* (TPDE-NET). Additional components were built which were specific to the AH-1W CDNU. The prototype developer selected components from these palettes and placed them on the canvas, thus creating an instance of the selected component. The prototype developer then customized the properties using custom editors, and connected the component instances by connecting them to the event handlers of other components. This technique was used to build a scaled down CDNU part-task trainer. This included a distributed simulation environment, familiarization routines, student model server, simulation server, and an instructor utility. Each of these processes will be described in detail later in this document.

The rapid prototyping techniques demonstrated with the proof-of-concept prototype clearly depicted their effectiveness at building part-task trainer prototypes. The methods used partially automate many of the repetitive steps involved with building new part-task trainers. The proof-of-concept prototype also demonstrates the high-level of reusability of the software components developed. Many of the same software components are reused in different aspects of this project. Finally, the maintainability of part-task trainers developed using these techniques is illustrated. A change to an actual CDNU device would in most cases only require the prototype developer to modify the model of the CDNU in one location. All of the ITS and computer networking code does not need to be touched. Also, if this component is used in several applications, then all changes to the CDNU can appear instantly in each of these applications. That is, we don't need to explicitly modify each application which uses the CDNU. Since all of these applications refer to the same component, all we have to do is modify this single component.

3.3 Distributed Architecture

One of the desired objectives of the Phase I prototype is to demonstrate the benefits of a distributed architecture for the part-task trainers. To accomplish this, we employed a simple distributed object model in the architecture of the Phase I prototype. The principle advantages of a distributed architecture that we want to illustrate include:

- **Distributed Simulation Environment**
- **Team or Group Training**
- **Centralized Student Model Database**
- **Instructor Utilities**

The distributed architecture of the Phase I prototype consists of four different types of processes which can execute on any machine within a network (note, this may be the same

machine or separate machines). A *student model* server provided a centralized database for student models, thus allowing mobile users to take full advantage of the student modeling capability and ITS from any location. A *simulation server* coordinated a distributed simulation environment for the students. The part-task trainer clients have the ability to connect to the simulation server and thus enter the student into the distributed simulation. An *instructor utility* demonstrated a second advantage of centralized servers. It enabled instructors to closely monitor the student's progress, possibly from a remote location. It also demonstrated how an instructor might intervene with a training sessions, if necessary. Figure 3 below illustrates the operating context of this architecture.

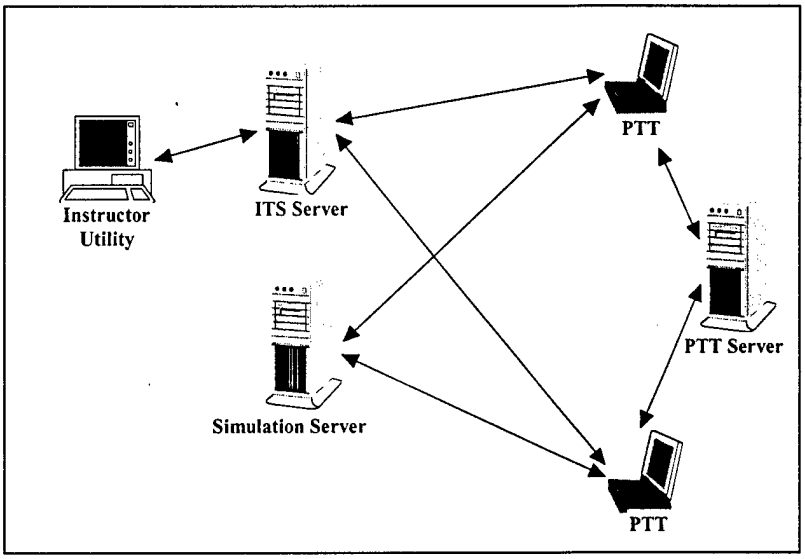


Figure 3. The distributed operating context of Phase I prototype

The above architecture was implemented using the Java Remote Method Invocation protocol. Distributed object protocols such as JavaRMI enable the programmer to refer to remote objects in nearly the same manner as they refer to local objects. This effectively abstracts many of the details of computer networking from the programmer. It also enabled us to support the use of distributed application architectures within a rapid prototyping environment. For example, in order to enable an ITS component instance to use a remote student model server, the prototype developer simply instantiates an instance of the student model server client component and connects it to the ITS component instance. The prototype developer doesn't need to know about the details of the computer network to successfully perform these tasks.

Despite the usefulness of the distributed architecture, it is import to recognize that a computer network may not always be available. Many of the overseas and ship-based training facilities have very few, if any, computer networks. To accommodate this, we were sure to implement the Phase I such that it could operate just as effectively in a standalone environment.

3.4 Intelligent Tutoring System

In addition to illustrating how part-task trainer prototypes could be built using the rapid prototyping techniques developed, we also wanted to demonstrate the benefits of integrating an intelligent tutoring system with a part-task trainer. This includes demonstrating how an ITS can be built using the rapid prototyping techniques developed and how the resultant part-task trainer will be a more effective training tool. To demonstrate these benefits, we once again turned to the domain of the AH-1W CDNU. We designed a simple intelligent tutoring system for training the CDNU navigation principles listed in *Table 1*. This ITS employed the general architecture illustrated in Figure 1.

The task of building the proof-of-concept intelligent tutoring system for the navigational procedures of the CDNU involved several sub-tasks. We needed to develop methods for building intelligent tutoring systems using the rapid prototyping paradigm. More specifically, we needed to design and implement reusable software components for building an ITS. We then needed to use these component oriented programming techniques to build a proof-of-concept part-task trainer for the CDNU. This involved defining a principle hierarchy, creating the interactive simulation, creating a scenario case-base, creating a remediation case-base, and creating remediation activities. Notice how each of these tasks corresponds to a module, or box, from Figure 1.

One of the most important parts of an ITS is the principle hierarchy. The concepts selected for this part-task trainer (see *Table 1*) were relatively straight-forward, so the principle hierarchy is not too complicated. Also, the focus of this portion of the research was on integrating intelligent tutoring systems with part-task training. Because of this, more time was spent on fine tuning techniques and less on the content of the ITS. Nonetheless, a fairly comprehensive principle hierarchy was developed for this project. This principle hierarchy is shown within a custom property editor in Figure 4.

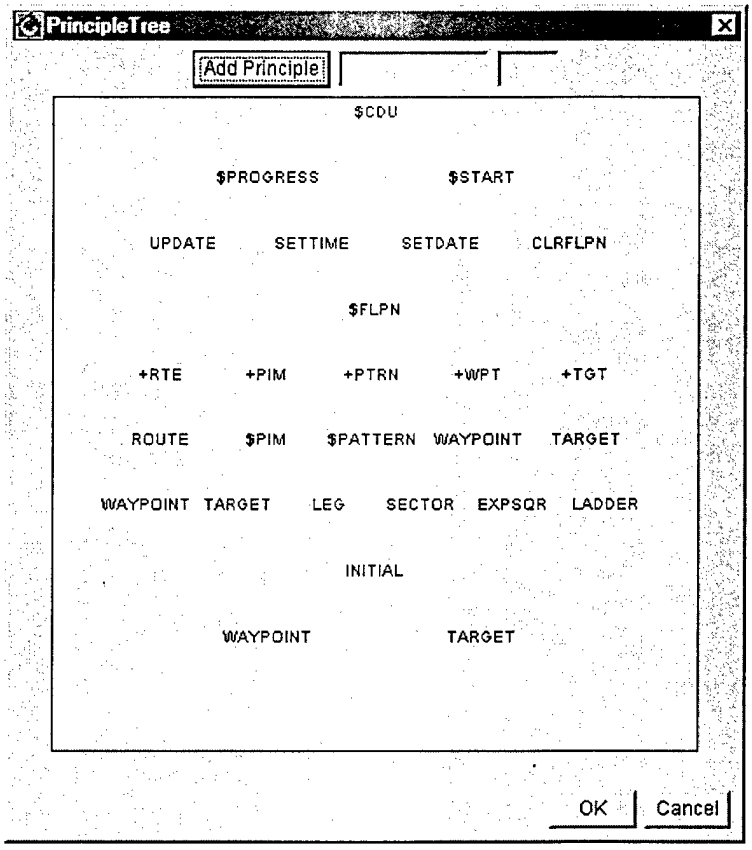


Figure 4. Principle Hierarchy for CDNU ITS

A *principle hierarchy* is used by the ITS to determine the relationships between the various principles involved in the training domain. In Figure 4, for example, the concept of *Route* is dependent upon the understanding of the *Waypoint* and *Target* principles. The ITS uses these relationships to determine which scenario or remediation activity to present to the student next. That is, if the student understands the waypoint and target principles, then the ITS will present the student with a scenario involving routes. The student will not be presented with a

scenario involving routes until he or she demonstrates an understanding of the target and waypoint principles. *A student model database* keeps track of the principles the student understands, fails to understand, or has not yet seen. The student model also keeps track of the scenarios and remediation activities the student has experienced. The instructor utility presented in section 3.5 provides a mechanism for browsing the student model information stored within the student model database.

Principle hierarchies can be constructed by instantiating a principle hierarchy component and then utilizing the custom property editor (shown in Figure 4) to build the graph of principles. Special characters within the various principles have special meanings to the ITS algorithm. For example, the '$' in the "$FLPN" principle signifies that the $FLPN principle is considered understood when all of its sub principles are understood. This is as opposed to the "ROUTE" principle, which is a principle in itself which is also dependent upon the understanding of its sub-principles. Note, these types of semantics could have been avoided by creating a more intricate principle data structure. However, these semantics were found to be a useful shortcut for the short amount of development time for implementing the Phase I prototype. The Phase II system will be more thorough in its definition of a principle.

The scenarios were selected by comparing the principles the student understands with the principles exemplified by scenarios within a scenario database. The construction of the scenario database was done graphically through the use of scenario and task components. The prototype developer builds a scenario by creating a scenario component and connecting a network of tasks and sub-tasks to that scenario. Each task includes a *description*, which describes the actions which the user must perform; a set of *parameters*, which are used to determine if the student successfully performed the task; and a set of *principles*, which signify the concepts the task exemplifies. The network of tasks determine the dependency between the various portions of the scenario. A simple example of a scenario is shown in Figure 5.

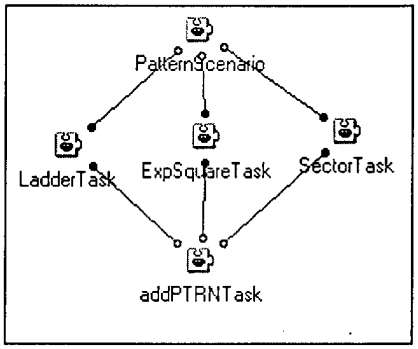


Figure 5. Simple Scenario from Phase I Prototype

In this scenario, the user must complete the *LadderTask*, *ExpSquareTask*, and *SectorTask* before he or she will be asked to perform the *AddPTRNTask*. This is a graphical way of signifying that the student must understand all of the basic pattern principles before learning how to add a pattern to a flight plan. These types of statements can be used to build extremely intricate scenarios and an extensive scenario database. Examples of this are presented in the Appendix.

Other major portions of the ITS from the Phase I, include a student model database and a student model server. The *student model database* provided a centralized location for the storage of student models. These student models were distributed to the intelligent tutoring systems over a computer network via the student model server. The *student model server* enabled a student to use any part-task trainer connected to the network. Whichever part-task trainer the student logs

on to would access the centralized student model server to acquire up-to-date information about the student.

The final major ITS component from the proof-of-concept prototype are the familiarization pages. *Familiarization pages* are a multimedia presentation of the course information. These familiarization pages are used as the principle remediation activity suggested by the ITS. They may be direct multimedia presentations of the principles or they may be plain text instructions such as *review pages 179-200 in the NATOPS manual* or *go see instructor Y.* Screen shots and examples of these ITS components are presented in the Appendix.

The development of the ITS portion of proof-of-concept prototype was a success. The experience not only resulted in an effective demonstration tool, but also served as a testbed for experimenting with the different ITS concepts involved. This experimentation paid off in the Phase II design of these ITS components.

3.5 Simulation Environment

A significant amount of development time was spent designing and developing the CDNU simulation environment. This portion of the Phase I prototype is extremely important for several reasons. First off, the simulation environment is perhaps the most visible portion of the prototype. Also, the simulation environment is where the majority of the learning will take place. We had several objectives for the simulation environment of the proof-of-concept prototype. One objective was to model the CDNU navigation procedures as precisely as possible within the amount of time available for development. Also, we wanted to provide some forms of engaging multimedia feedback to the student. This is one of the identified limitations of current part-task trainers and we wanted to demonstrate that it could be overcome efficiently. Also, we wanted the simulation environment to be distributed. A distributed simulation will aid in the students experience by making the simulation more interesting. Also, a distributed simulation could potentially be used for team training. Finally, the simulation environment needed to have the ability to run the scenarios defined in the ITS components and provide hooks, such that the ITS components could monitor the student's performance within the simulation environment.

The simulation environment which was developed, involved a handful of software components. One component included a software model of the CDNU along with a graphical mock up of the device. This component enabled the user to perform many of the CDNU navigational procedures within the simulation environment. In addition to being used within the simulation environment, this component was used within several familiarization pages. Another component developed is a multimedia map component. The multimedia map component played a significant role in the simulation, by engaging the students interest and providing multimedia feedback to his or her actions. Additionally, the map display component depicts the platforms of every student within the distributed simulation. This further enhances the user experience and alludes to how the part-task trainer may be used for team training. Additionally, other simple instrument components were created to provide access to important information and feedback to the student. These instruments included speedometer and compass. Controls for entering a distributed simulation, modifying the simulation's settings, and logging on to the ITS were also created. Additionally, message window components were created for displaying ITS-specific messages and for displaying general simulation messages to the student. Finally, several non-visual components were developed to handle the internal logic of the part-task trainer. These non-visual components included platforms, a timer for synchronization, modules for doing navigational computation, and several others. These components were linked in with the ITS and distributed simulation components. Screen shots of the development environment and the end part-task trainer can be found in the Appendix.

The simulation environment developed attained its goal of providing a multimedia-rich model of the CDNU and various feedback components. Because the development of the simulation environment followed the component oriented programming event model, it was extremely straight-forward to integrate with the ITS. To integrate the ITS, all we had to do was enable the ITS to catch the simulation events and determine if they match any of the tasks currently assigned to the student. The ITS used a message window to pass information to the student and to request the student to perform specific tasks.

3.6 Instructor Utility

A simple instructor utility was developed to demonstrate one of the advantages of the distributed architecture. This utility enables instructors to monitor a student's progress from a remote location. The instructor utility simply connects to the central student model database and displays up-to-date information about the student's mental model. This information includes a color coded graph of the principles the student has achieved, missed, or not yet seen, as well as a list of scenarios the student has seen. Other possible uses of an instructor utility would be to allow the instructor to modify the principle hierarchy or the student model in some way. Also, the instructor may suggest which scenario the student should perform next or even interact with the student in some way through the distributed simulation environment.

The instructor utility developed for the proof-of-concept prototype used many of the same software components as the CDNU part-task trainer prototype. This property enabled us to implement the instructor utility in a very short amount of time. Screen shots of the proof-of-concept instructor utility can be found in the Appendix.

4. Phase II System Design

This section provides an overview of the Phase II Trainer-PDE system design developed during the Phase I research.

4.1 Operational Trainer-PDE Prototype

The Trainer-PDE system is a rapid prototyping environment for building part-task trainers. The resultant part-task trainer prototypes implemented with this system will incorporate intelligent tutoring systems to cater training programs to specific individuals. The environment and resultant part-task trainer prototypes will take advantage of Internet/intranet technologies to enhance portability and enable the part-task trainer prototypes to take advantage of computer networks. Additionally, the system will be capable of running as a standalone system when no network is available. The prototyping environment will take advantage of open standards, component models, and commercial-off-the-shelf software. This awareness of software engineering trends will enable the Trainer-PDE system to maximize reuse and provide an environment to efficiently prototype part-task trainers.

Many inter-related concepts are involved in the Trainer-PDE system. A thorough understanding of each of these concepts is required for designing and implementing the Phase II operational Trainer-PDE prototype. These concepts include:

- **Part-Task Trainers (PTT)**
- **Distributed Part-Task Trainers**
- **Rapid Prototyping Environment**
- **Intelligent Tutoring Systems (ITS)**
- **ITS Authoring Tools**
- **Component Models**
- **Distributed Object Models**
- **Visual Programming Environments**

4.2 Trainer-PDE System Overview

This sub-section describes the high-level system overview for the Trainer-PDE system. This includes the major components of the system and how these components interact to provide an effective tool for prototyping part-task trainers. The next section will describe the system architecture at a more detailed level.

The system will be comprised of a set of software components designed for use within a COTS tool to build part-task trainer prototypes. These components will be divided into palettes based on their functionality. These palettes of components are imported into the COTS tool and used by the developer to build part-task trainers. Figure 6 illustrates how these components might appear within a COTS visual development environment. Notice how this roughly resembles the screen shots from the Phase I prototype, in the Appendix. This is because the proof-of-concept prototype followed a very similar design as the one presented here. Essentially, the proof-of-concept prototype is truly a scaled-down version of Trainer-PDE. This is as opposed to many proof-of-concept prototypes which simply provide a visual mock-up of what an end system will look like, without modeling any internal behavior.

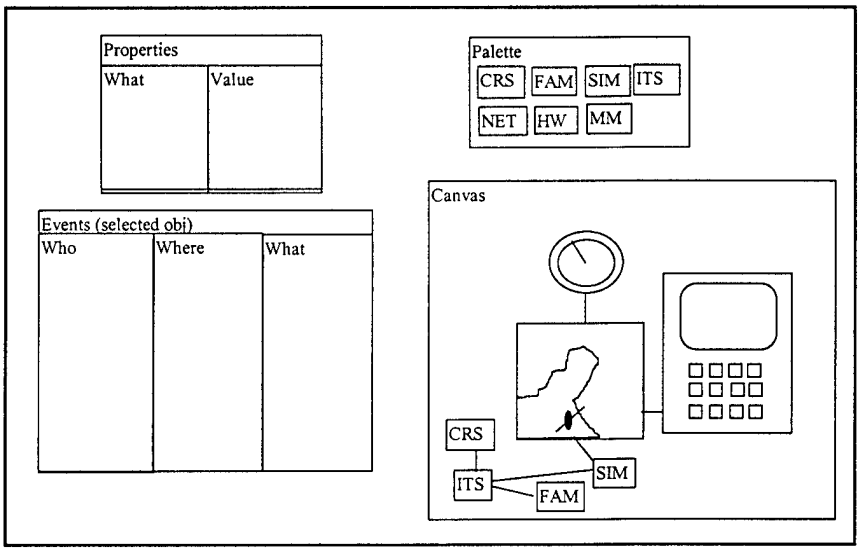


Figure 6. Rapid Prototyping Environment within a COTS tool

The palette provides an interface for selecting a component to use in the part-task trainer prototype. These components are placed on a canvas and visually manipulated using custom property editors and event editors. Figure 7 illustrates how these component instances might appear on the canvas. This particular figure represents how the components would be used to build a part-task trainer for the CDNU. However, other part-task trainers will have a very similar architecture. The major differences will reside in the simulation components of the part-task trainers. Property editors and events provide mechanisms for describing how the component instances behave and how they will interact with one another. Using the COTS tool to graphically connect component instances on the canvas will describe the directions that the events flow. Selecting a connection will bring the user to an event editor where the user can formally describe the information passed between the components. Selecting an instance of a component will bring the user to a property editor where the user can specify values for the components member variables. The functionality described is provided by COTS tools which support the component model used to build the Trainer-PDE components. For example, if JavaBeans is used to implement Trainer-PDE, then IBM Visual Age for Java or Sun Java Workshop may be used to connect and edit the components. The Trainer-PDE system will be

composed of implementations of components, custom events, custom event editors, and custom property editors, all geared towards rapid prototyping of part-task trainers. Additionally, templates and roadmaps will be provided to ease new developers into the rapid prototyping paradigm.

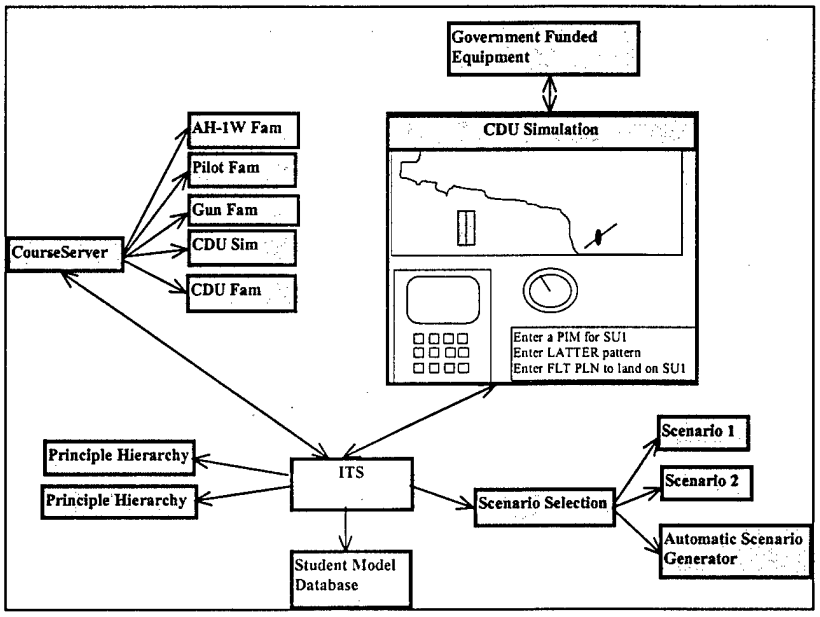


Figure 7. Rapid Prototyping Environment

As a final note on Figure 7, notice how the physical layout of the components represented by shaded boxes, and the event connections represented by arrows, closely resembles an architectural design for the part-task trainer as it might appear in an object modeling tool such as Rationale Rose. However, unlike an object modeling tool, these boxes are actual component instances and with a click of a button, the application can be executed. This is rapid prototyping - the bridging of the design, implementation, and testing phases of prototype development.

4.2.1 Components

The reusable software component will form the foundation of the Trainer-PDE system. A software component is defined by a software object which conforms to some standard interface. This standard interface allows the public member variables and public methods of the component to be realized at run-time, thus enabling their use within a visual development environment. There are many standard component models and many COTS visual development environments which support these component models. The majority of the tasks for implementing the operational Trainer-PDE prototype will include implementing the sets of software components for building distributed part-task trainer prototypes, defining the custom events which enable these components to interact, and developing the custom editors for defining and manipulating component properties and events.

4.2.2 Palettes

The Trainer-PDE System will be divided into palettes based on the functionality of the components. These palettes will include the following: course structure (TPDE-CRS), intelligent tutoring systems (TPDE-ITS), multimedia (TPDE-MM), familiarization (TPDE-FAM), simulation (TPDE-SIM), network (TPDE-NET), and hardware (TPDE-HW). The components of these palettes will be edited and linked together within a COTS visual prototyping environment.

Each component is defined by a set of properties, set of events it can handle, and a set of events that it fires. Custom property editors and custom event editors will be developed to edit the components properties and events.

4.2.3 Event Model

The components will communicate with one another using custom events. Each component specifies the types of events it can send and the types of events it fires. Information will be passed between components through these custom Trainer-PDE events. Components will respond to these events based on the information they pass. Each component may define its own type of events and each component can define its own set of event handlers for existing events.

Simulation components, for example, will define several events for passing information amongst objects within the simulation. Intelligent tutoring system events will be used for passing information amongst the ITS objects. A simulation event titled *task event* will be fired every time the student performs some task within the simulation. The task event will pass along information about the task in its parameters. The task type *waypoint_entered* and the parameters *lattitude_x* and *longitude_y* may be passed as the task parameters. The ITS component instance will be specified to catch these events. When these events are caught by the ITS component, the ITS component will check to see if they match a current task assigned to the student and whether the student performed the task correctly. This information will then be used to update the student model.

There are many other examples of how events will be used. Some of these include a *timer event,* which will fired by a *clock* component instance every so many milliseconds, a *platform event,* which will be fired by the *platform* component every time the platforms state changes, an *update distributed simulation event,* which will be fired by the *distributed simulation server* component every time the distributed simulation needs to be updated, and many more.

The use of an event model to connect components together is a very robust mechanism for defining component interaction. The implementation of each component can change its internal structure without having to explicitly update the components it interacts with. Components catch and handle events and components throw events; there is no direct connection between the components. One does not even need to know the properties or methods that the components implement to use the components. This significant specialization of the object oriented paradigm is a key property for rapid prototyping environments such as Trainer-PDE.

4.2.4 Property Editors

Each component is defined by its properties and the events it fires and handles. To use object oriented terminology, component properties are essentially the public member variables of software components. These properties will be defined by the prototype developer through standard and/or custom property editors. Standard property editors are editors defined in the component model standard and implemented in the COTS tool that supports that standard. These standard property editors are appropriate for defining simple data types such as strings or integers. More complicated data structures will be defined using a custom property editor which is developed specifically for that data structure. A significant task in the development of the Phase II system will be designing and implementing these custom property editors. The careful design of the user interface for these custom property editors will enhance the visual development capabilities of the rapid prototyping environment. In cases where the property is a simple data type, then a default property editor will be fine. However, if the data type is a graph or something more complicated, then a custom property editor should be used. Figure 4 is a screen shot of a property editor from the phase I prototype. This property editor is used to define

intricate principle relationships and thus, capture domain knowledge for an intelligent tutoring system.

4.2.5 ITS Architecture Template and Development Roadmaps

Because most ITS enhanced part-task trainer prototypes developed with Trainer-PDE will have a similar architectural design, templates and road maps will be provided to facilitate the prototyping process. Templates will lay the foundation for the prototype by providing the base set of components and event connections that are common to the type of prototype under development. A roadmap will be provided along with the template. This roadmap will describe how to edit the property values of the components and subsequent steps to proceed with the prototype development. The template will closely resemble Figure 7 when viewed within the supported COTS tools. Because templates are not defined in most object models, it is likely that the templates will have to be developed for each COTS tool we wish to directly support. This, however, will not be a significant task. The challenging task will be implementing the components, events, and editors to be used within the templates. Documents for building these templates within new COTS tools will be provided, in case the chosen environment is not supported.

4.3 Phase II System Architecture

This section describes a detailed design for the part-task trainer rapid prototype development environment.

4.3.1 Platform Specification

There are several platform requirements for this system. The part-task trainer prototypes will run on a number of platforms in a standalone manner or across the Internet or an intranet. The part-task trainer rapid prototyping environment will run at least on a Windows 95/98/NT platform, but should be trivial to port to other platforms. The environment will be a COTS tool making use of the components this document describes. If a platform-independent component model is chosen (see next section) then the rapid prototyping environment may be used on several supporting platforms.

4.3.2 Development Environment

The system will be built using an open standard component model such as JavaBeans, the Common Object Request Broker Architecture, or the Distributed Common Object Model. By following an open standard, our system may be integrated with many existing COTS visual development environments which support the chosen component model. Additionally, there are many tools which allow these component models to work together. The decision as to which component model to choose for system development is an important one, however, it is also important to remember that different component models can work together and it is important to remember that one does not need to limit himself to just one. Nonetheless, the more care taken in selecting the component model, the more powerful the resultant system will be.

The current recommendation would be to use the JavaBeans component model in concert with the CORBA distributed object model. Java Beans is the most widely supported component model and works very well with Internet/intranet applications. However, despite recent advances, there are still some questions regarding efficiency to be answered. These should be evaluated early in the Phase II. Using CORBA as a distributed object model is one way of possibly addressing these issues. CORBA is a platform and language-independent object model and as such, more efficient languages may be used for objects which require the efficiency of a lower level language, such as C++. Additionally, the use of CORBA with Java is

gaining a lot of industry momentum. Already, tools exist which facilitate the use of these two concepts together. Furthermore, CORBA has been selected by Sun Microsystems, the maintainers of the Java programming language, as the underlying distributed object model to be used within the JDK version 1.2. This will further insure the longevity of these two standards. However, things change very rapidly in this industry and these issues should be revisited come implementation of the system. The main advantage of DCOM, currently, is that it is slightly more efficient than CORBA and JavaRMI. The main disadvantage is that there is a higher learning curve involved and there is very little platform portability.

4.3.3 Component Interaction Diagram

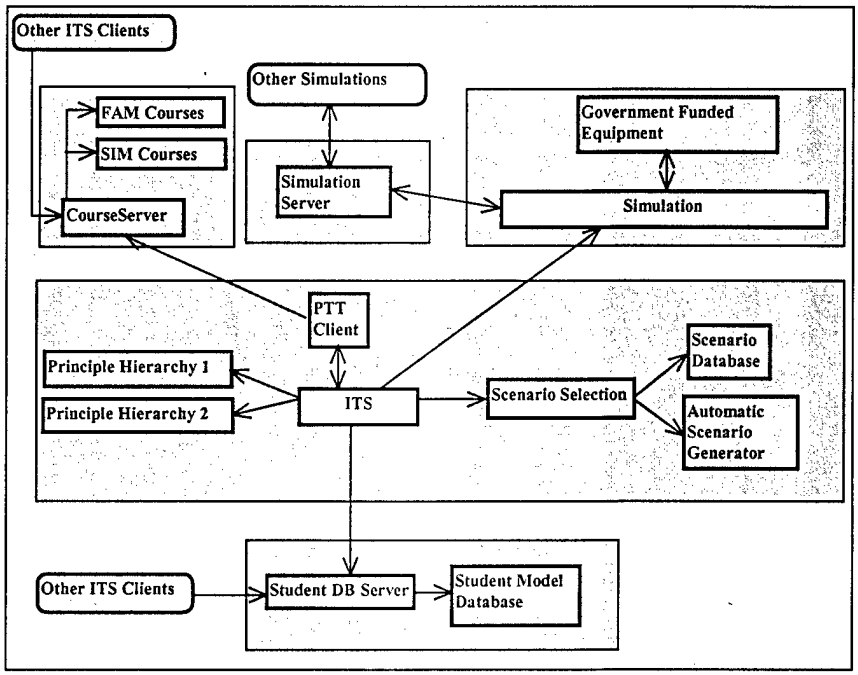


Figure 8. Component Interaction Diagram

Figure 8 is a component interaction diagram for a typical part-task trainer prototype developed using trainer-PDE. The dark gray boxes represent a single process running on some machine while the labeled boxes represent the software components which define those processes. Each of these processes may run on a different machine or all on the same machine. The labeled boxes with rounded corners represent duplicate processes interacting with the same components. Several ITSs and Simulations may be using the same course, students, and simulation servers at the same time, for example. The inter-process communication is illustrated by the arrows. For example, the arrow directed from the *Student Database Server* component to the *Student Model Database* component illustrates that the database server uses the student model database, but not vice-a-versa. The inter-process communication (IPC) which is used, works within a distributed system as well as on a standalone machine. This IPC is facilitated through the adherence to a distributed object model. The diagram (Figure 8) only depicts two principle hierarchies, however, the system will support an arbitrary number of principle hierarchies. As a final note, some of these components may actually be made up of sub-components.

The first question which will likely come to mind is, if all part-task trainer prototypes are going to follow this *architectural design template*, why do we need components? Or, for that matter, why do we need templates? The answer is that each one of these components will be

specialized into one or more part-task trainers or utility applications. When there are several uses for specialized components, the software may be reused simply by creating new connections between the components. For example, Figure 9 illustrates several ITS client components sharing a Principle Hierarchy component.

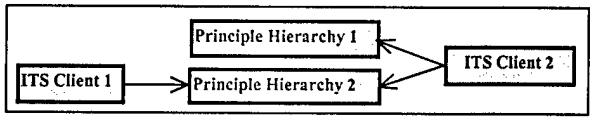


Figure 9. ITS Clients sharing a principle hierarchy

Templates are useful because even though most part-task trainers developed with this system will follow a similar design, there will be subtle differences. The templates will provide a starting point for the part-task trainer prototyping process. For example, the part-task trainer developer may want to add additional principle hierarchies or reuse components from an earlier part-task trainer prototype. These types of decisions will be domain-dependent and must be made by the part-task trainer prototype developer.

4.4 Trainer-PDE Software Components

The development of the components in Figure 8, including the development of custom property editors, events, and event editors, will be the most significant effort of the Phase II implementation. A brief description of each of these components follows.

Course Server Component:

The Course Server component is designed to coordinate students' sessions with the Part-Task Trainer. This component will reside on a centralized server machine and will serve the appropriate course sections to the part-task trainers as needed. For example, when a user first uses a part-task trainer, he or she will likely be presented with a course overview section. Once the student is finished with this section, the Course Server will direct the Part-Task Trainer Client component to load up an ITS Course section on the material covered.

In effect, the course server will direct the student through a graph of course sections. This is coordinated by communication between the Part-Task Trainer client, which coordinates activity on the host machine, and the Course Server component. A sample graph is illustrated in Figure 10. Each box represents a course section which may be an application, a web page, or a collection of web pages or applications.

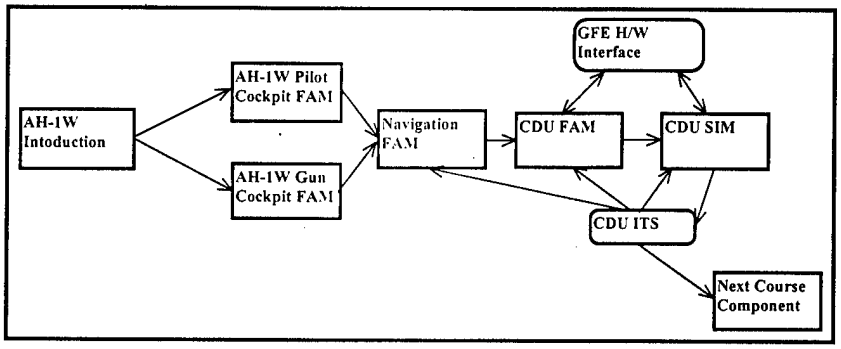


Figure 10. Course Flow Diagram

FAM Course Component

The FAM (or Familiarization) Course components represent sets of familiarization applications or pages. These components contain information as to how to invoke a familiarization course section. This may be a web address or execution statements. These

components should also describe the platform requirements for the course section. For example, the host machine must have Java Runtime 1.1.6 or higher in order to load this course section. Familiarization course sections are used to present the knowledge to be trained. A sort of multimedia textbook for the training domain.

SIM Course Components

The SIM (or Simulation) Course components represent sets of simulation applications or pages. These course sections are very similar to the FAM Course components, but are used to present simulation course sections as opposed to remediation course sections.

PTT Client Component

The Part-Task Trainer Client component is responsible for coordinating part-task training activity on a host machine. This component communicates with Course Server components and loads or executes the appropriate course sections on the client machine. This component also communicates with currently executing course sections and passes along relevant information to the course server.

ITS Component

The ITS component encapsulates the intelligent tutoring system algorithm and control structure. Instances of this component will reference the supporting data structures for the particular ITS as well as an Interactive Simulation component and Part-Task Trainer Client component. This component will reference *n* Principle Hierarchies which maintain the expert knowledge for the domain. The ITS uses this expert knowledge to build student models and select appropriate scenarios and remediation activities. The ITS refers to the Student Database Server component to access and update information about the students. The ITS refers to Scenario Selection components when it needs to select a scenario for the Student. The ITS will catch events fired by the Simulation component. The ITS will examine these simulation events to see if they match current tasks assigned to the student. This is how the ITS will measure the students performance within the Simulation. Finally, the ITS refers to the Part-Task Trainer Client component when the student is finished with an ITS session.

Principle Hierarchy Component

Instances of the Principle Hierarchy component are used to maintain the relationships between the principles for the training domain. This data structure is where the majority of the expert knowledge is stored. A clever user interface (e.g., Figure 4) will ease the task of building the expert knowledge. These components are primarily used by ITS components, but may also be used by instructor tools or other utilities.

Scenario Selection Component

The Scenario Selection component is responsible for selecting an appropriate scenario based on a student model. This component references Scenario Database components and Automatic Scenario Generator components. The component is used primarily by ITS components. The ITS component will request a scenario which emphasizes a set of principles. The Scenario Selection component will then search the scenario databases and generators that it references (this may be more than one) and returns the most applicable scenario.

The artificial intelligence technique known as case-based reasoning is used by the selection component to select the scenario. *Case-based reasoning* examines a database of historical problems and solutions, attempts to find a situation which resembles a current problem, and then attempts the historical solution to the current problem. Case-based reasoning is a powerful AI methodology which has been successfully applied to intelligent tutoring systems by SHAI.

Scenario Database Component

The Scenario Database component collects a set of scenarios into a database. These scenarios are indexed by the principles they emphasize. This component is referenced by the Scenario Selection component when a new scenario is needed. This component may also be used by Instructor Utilities to add, edit, or remove scenarios from the database.

Scenario Generator Component

The Scenario Generator component is responsible for automatically generating a scenario which emphasizes certain principles. This component will be referenced by a Scenario Selection component and will be called on to generate a scenario when there are no appropriate scenarios within the scenario database. This component will likely actually be an *Interface* or *abstract class* which will need to be defined when new simulations are created. Interfaces and Abstract classes are data structures whose interfaces are defined, so other components may reference them, but with no, or very little logic built into them.

Student Database Server Component

The Student Database Server component is used to provide access to remote student model databases. This *middleware* contains the logic for handling requests from ITS Clients, passing them on to the appropriate databases, and returning or updating the student model information. Having a centralized student database enables highly mobile students to use any part-task trainer on a network and the part-task trainer will go to the central location to find the student model for that student. This component may be used by Instructor Utilities as well as ITS components.

Student Model Database Component

The Student Model Database component maintains information about student models. These fields are listed in Table 2. This component may be used by the Student Database Server to distribute the information to remote intelligent tutoring systems or other utilities. The component may also be used directly by an intelligent tutoring system or other utility to provide the information locally.

Table 2. Student Model Database Fields

Student ID	Unique identifier for each student
Principles Seen	Each principle the student has seen
Principles Achieved	Count for achieving each principle
Principles Missed	Count for missing each principle
Scenarios Seen	List of scenarios seen

Simulation Component

This component provides the interface between the interactive simulation and the ITS, as well as the interface between the interactive simulation and the distributed simulation server. The internal components which are used for building simulation environments for part-task trainers will, in most cases, need to be specialized for different part-task training domains. As more part-task trainers are built using the Trainer-PDE environment, more multimedia components will be added to the library. This will ease the process of building new specialized components for new types of simulations. The Simulation component described here will provide a standard interface between the custom simulation components and the rest of the part-task trainer.

Simulation Server

This component will coordinate distributed simulation environments for part-task trainers. If a student using a part-task trainer wishes to enter a distributed simulation, the local

simulation component will look up this server component. The server component will then receive all relevant updates from the clients and pass them on to every part-task trainer participating in the distributed simulation. Simulation servers could also be used to coordinate team training environments or student/instructor interaction.

In addition to the components listed above, general components for building multimedia web pages and interactive simulations will be integrated into the environment. There are many third party tools which provide a rich set of software components for building multimedia environments. These multimedia environments will provide a lot of the functionality we need for building interactive simulations and multimedia remediation activities. We will, however, undoubtedly want to implement our library of multimedia components for building simulations and remediation screens in addition to reusing these third party components.

4.5 Phase II System Design Summary

The System Design presented will result in a fully operational Trainer-PDE system. In Phase II, this system will be used to build two part-task trainers for the Navy (see Section 5.1). The Trainer-PDE rapid prototyping techniques are leveraged upon the technologies of distributed object models and component oriented programming. The Trainer-PDE computer-based training techniques are leveraged upon the technologies of intelligent tutoring systems, student modeling, and case-based reasoning. The result of the end Trainer-PDE system will be the ability to rapidly and efficiently build highly effective and flexible part-task trainers. These part-task trainers will have the ability to take advantage of network resources, incorporate state-of-the-art training techniques, and will be much easier to maintain and modify than traditional part-task trainers.

5. Future Work & Commercialization

This section outlines the future of the Trainer-PDE project and the eventual commercialization objectives.

5.1 Phase II Technical Objectives

The primary objective of the Phase II effort will be to enhance Navy training and readiness by developing a system which can efficiently prototype highly effective part-task trainers. Two domains will be selected and an operational rapid prototyping environment capable of building fully functional part-task trainer beta prototypes for those two domains, will be developed. Part-task trainers developed with our Phase II system will integrate Intelligent Tutoring Systems capable of catering training programs to a specific individual's strengths and weaknesses. This property will enable part-task trainers to train more effectively and tackle new classes of training tasks. Additionally, we seek to reduce the cost of part-task trainer prototype development, delivery, and maintenance. By leveraging the Phase II technology on standard component models and distributed object models, this objective will be accomplished. This means emphasizing the interconnected concepts of open standards, software reuse, commercial-off-the-shelf software, virtual machine languages, and Internet/intranet technologies. Through the effective use of these concepts, we seek to partially automate the PTT prototyping process and thus reduce the cost of development and maintenance. Through the effective use of a distributed object model, part-task trainers developed with the Phase II system will reap the advantages of running in a networked environment as well as on a standalone system. These objectives are described in detail below.

The Phase II technical objectives have been endorsed by Major Wenrich of the Helicopter Training Squadron HMT-303 located in Camp Pendleton, CA. Major Wenrich coordinates the computer-based training (CBT) facility for the squadron. Our objectives include

a continued relationship with HMT-303 to better insure the quality of our work and the success of the project.

5.1.1 Improve Efficiency of PTT Development and Effectiveness of Resulting PTT

The Phase I proof-of-concept prototype demonstrated the feasibility of using component oriented programming to develop a new concept of rapid part-task trainer prototype development. Reusable software components were developed which could be visually manipulated within numerous COTS tools to develop part-task trainer prototypes. Techniques were developed to use this component oriented programming paradigm to build part-task trainers enriched with ITS concepts. The integration of ITS concepts with part-task trainers enables a part-task trainer to model a student's understanding of the training domain and to utilize this knowledge to tailor a training program to individual students. The components developed will be highly portable and will have the ability to run in a distributed environment, as well as on a standalone machine. Running in a distributed environment will enable part-task trainers trainees to interact with one another within a networked simulation, enable instructors to remotely monitor and interact with a students training environment, and enable the part-task trainers to be available to highly mobile students and instructors. In Phase II, two fully functional part-task trainers will be built using the rapid prototyping environment. Below, the sub-goals of this objective are enumerated.

A. **Enhance COP techniques for building interactive simulations, distributed training environments, multimedia, and ITSs:** As part of the Phase II effort, we will extend the techniques developed in the Phase I for using a component-oriented programming paradigm for the rapid prototyping of part-task trainers. This primarily involves identifying the key entities common to this new breed of part-task trainers and defining how they can be described within a reusable software component. This includes components for supporting the development of interactive simulations, distributed training environments, multimedia, and intelligent tutoring systems.

B. **Enhance integration of ITS concepts into PTT prototypes:** In Phase I, we successfully demonstrated the use of an Intelligent Tutoring System within a CDNU part-task trainer. The techniques developed for this task will be extended to support a wide breadth of part-task training domains. This will primarily involve further investigation into how detailed mental models of a student's understanding of specific domains can be built and represented within a part-task trainer.

C. **Reduce cost of PTT development:** We will implement an operational prototype of a component-oriented rapid prototyping environment to build two fully functional PTT beta prototypes. We will use the rapid prototyping techniques developed in Phase I to build the reusable software components required to implement two fully functional part-task trainers. By following open standards and emphasizing reusable software, this task will be accomplished. The rapid prototyping environment will take the form of the developed software components imported into a supporting COTS tool.

D. **Improve the Training and Readiness of Navy personnel in two PTT domains:** The Phase I prototype demonstrated the feasibility to use the developed rapid prototyping techniques to build an AH-1W CDNU navigation part-task trainer prototype. The Phase II effort will build on this accomplishment by utilizing these techniques to build two fully functional part-task trainer prototypes. A need has been expressed for a fully functional AH-1W CDNU communication and navigation procedure trainer. We will focus on this domain for the first part-task trainer. Additionally, we will evaluate the need for weapon systems

trainers (WST) within the H-1 community and select the second class of part-task trainer to implement.

E. **Enhance accessibility of ITS authoring:** It is often desirable to provide ITS authoring utilities intended for use by instructors with little or no programming experience. These utilities allow non-programmers to author scenarios and populate knowledge bases for specific intelligent tutoring systems. We will develop techniques for implementing such tools using the component-oriented programming paradigm and integrate them into the rapid prototyping environment. **(Phase II option)**

5.2 Commercialization Plans

There are numerous commercial outlets for this effort. The Phase II project will leave us with two fully functional part-task trainer prototypes as well as a rapid prototyping environment for implementing new part-task trainers. This means we will both have the ability to produce new part-task trainers at a relatively low cost and we will have a rapid prototyping environment which can be extended and sold to other part-task trainer developers. Further, the domain of distributed part-task training over LANs, intranets and the Internet is closely related to many other training domains. Distance learning, for example, is gaining a lot of momentum in academic, commercial, and government markets. Our tool could be modified to support those markets as well as similar markets within the Navy and other DoD branches. Further, the properties of this SBIR position the effort precisely into our company business plan of generalizing and marketing our previous work in Intelligent Tutoring Systems. More specifically, this project provides powerful ITS authoring abilities, particularly for distance learning systems and distributed simulation environments - two fast growing markets in the world of computer-based training.

A prime candidate for the described product and services is the Navy. The Navy has a very real need for effectively training personnel at a lower cost. The Navy would also be attracted to the distance learning and distributed training environments our part-task trainers will provide. In phase I, we worked with the H-1 helicopter training school, HMT 303, at Camp Pendleton. The goal is to better understand their training requirements and to determine how our tool would help them best. We would continue this relationship through a Phase II effort. The computer-based training coordinator for HMT-303, Major Wenrich, is also interested in continuing this relationship. By working closely with contacts within the Navy, the beta part-task trainers will be recognized as a valuable tool to training programs and will be adopted into Navy training programs. SHAI is currently following a similar track with a Navy Phase II project involving intelligent tutoring systems for TAO officers. The training tool will soon be adopted into TAO training at the Surface Warfare Officer School.

The likely commercialization avenue for the Phase II using this model is to generate enough interest that a Navy fleet instructor initiates a Fleet Operational Needs Statement (FONS). This FONS then generates a Mission Needs Statement (MNS), which in turn generates an Operational Requirements Document (ORD), and finally, a Program Objective Memorandum (POM), and thus a direct funding source. The POM for the project would be for 2004. However, seed funding through the Presidential Requirement (PR) program of 2003 could help carry the project through this process after the Phase II is complete.

Another prime route for commercialization is to use the tool to create ITSs for individual clients and markets. The core ITS concepts and the software components which realize those concepts can seemlessly be carried across domains. We would be able to use our software components and workbench to quickly build new classes of intelligent tutoring systems. Marketing Intelligent Tutoring System creation services is similar to SHAI's core business of

marketing Artificial Intelligence research and development services. SHAI has been very successful in marketing such services.

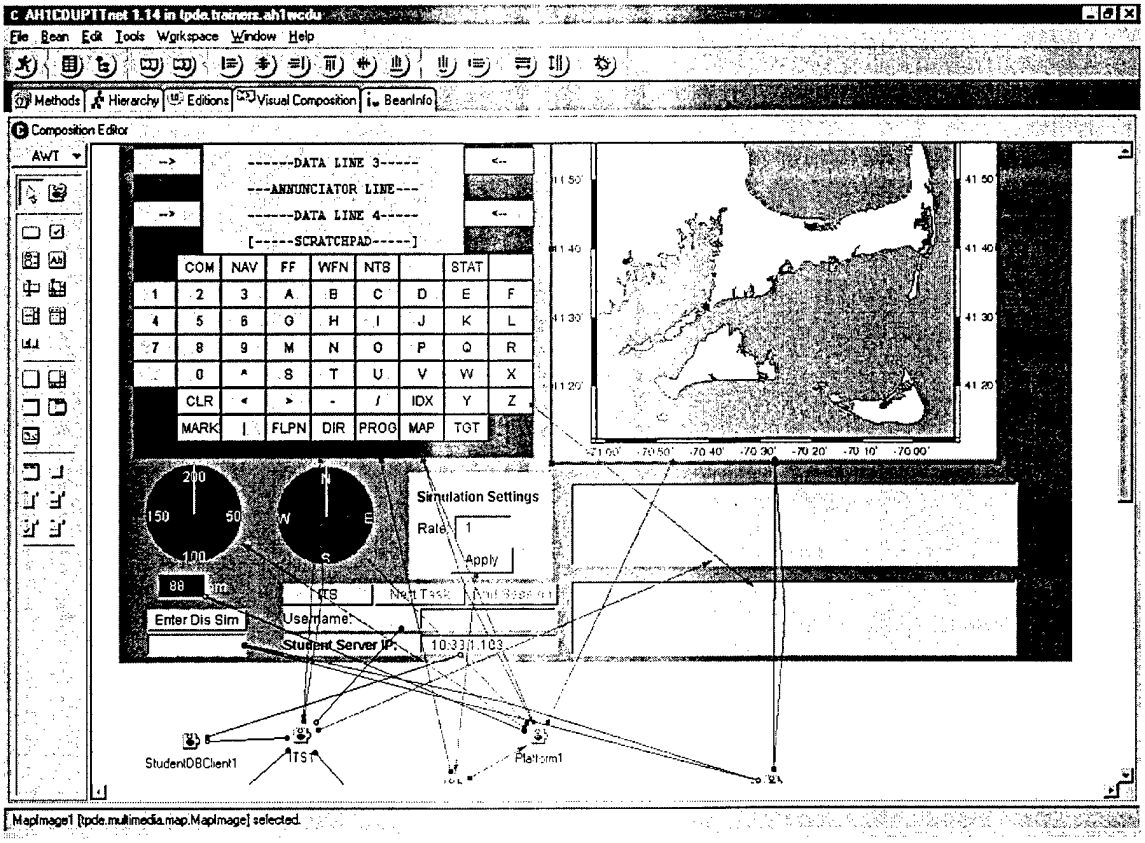
Marketing Intelligent Tutoring Systems to individual, vertical marketing segments will be highly dependent on the markets involved. For example, SHAI has developed relationships with several educational software companies and text book publishers. This could be used in preparation for intelligent tutoring systems aimed at high school and remedial college math (algebra, geometry, trigonometry, and calculus) and science (physics, chemistry, and astrophysics). Commercial aviation and corporate training are two prime potential markets for this research. Additionally, SHAI has solid contact with customers in the military likely to purchase developed ITSs and PTTs, including the Dismounted Infantry School at Fort Benning, the Armored Infantry School at Fort Knox, the AEGIS Program Office, the AEGIS Training Center, AEGIS Training Group, Surface Warfare Officer School, Army STRICOM, and SPAWAR. All of these angles will be explored in detail during the Phase II.

Appendix A: Screen Shots

This appendix provides a series of screen shots from the Phase I proof-of-concept prototype. Each of these screen shots relay a different concept presented in the final report.

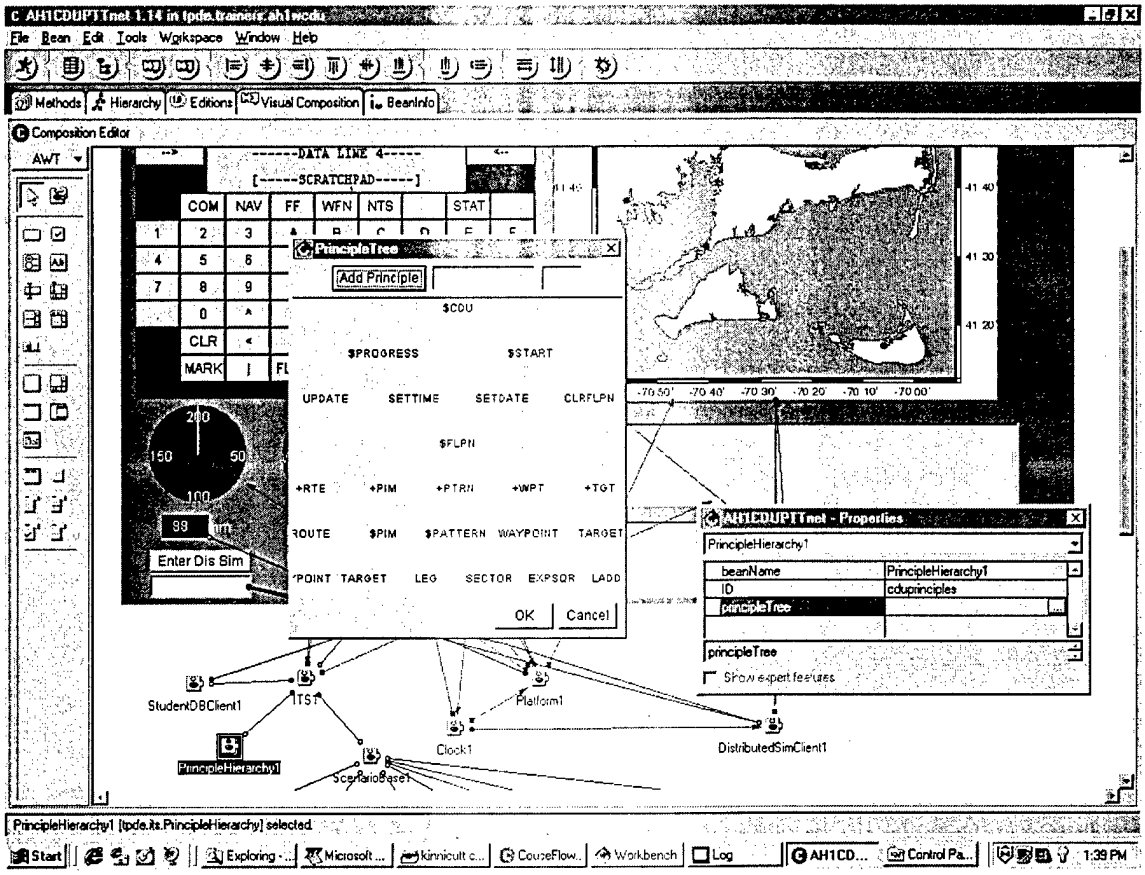
Part-Task Trainer Rapid Prototyping Environment

The Screen Shot below shows the reusable JavaBeans component from the Phase I proof-of-concept prototype being utilized within the IBM Visual Java v2.0 to build the AH-1W CDNU Part-Task Trainer. The portion of the environment which is seen in this screen shot focuses mainly on the simulation portion of the Part-Task Trainer. *Note* the component instance **Platform 1** near the bottom of the screen and how it is connected to the map image, CDNU, speedometer, and compass components. As discussed in the final report, the platform component fires events every time its state changes. When this happens, all of the connected components respond to these changes in different ways.



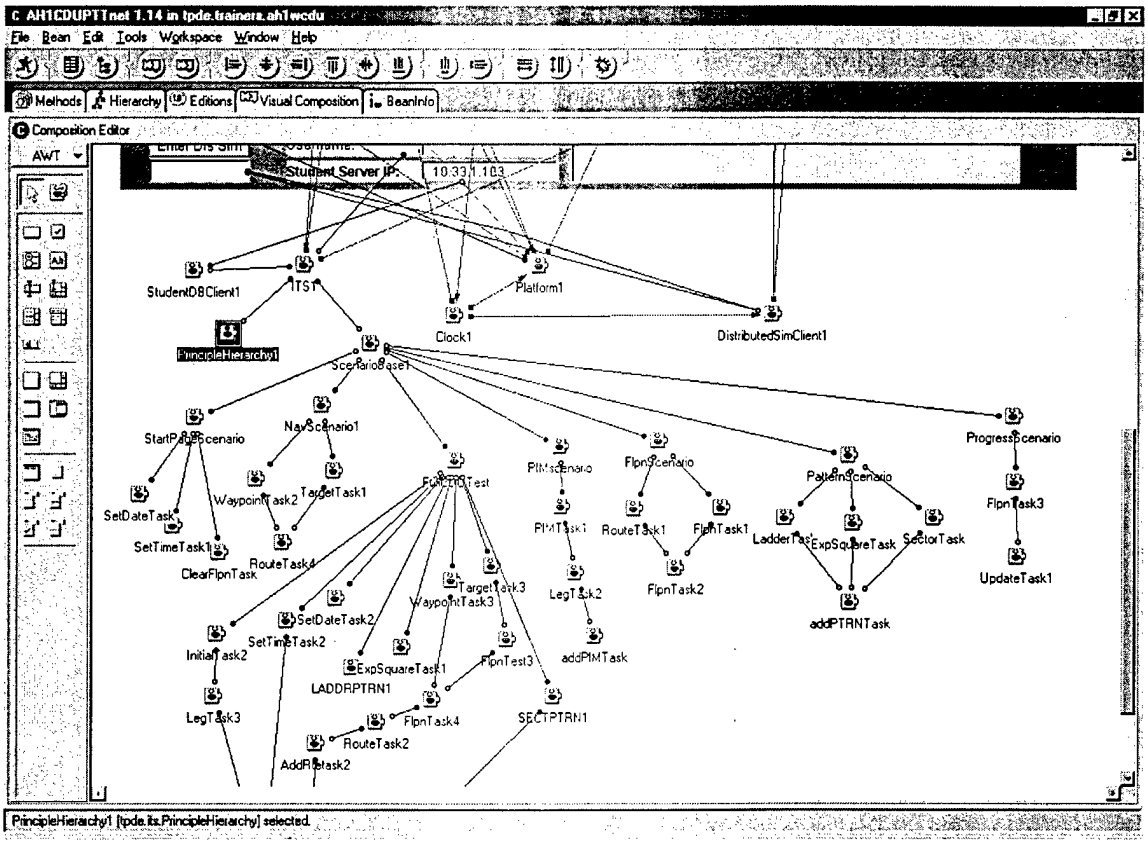
Property Editors

This screen shot is similar to the previous shot. However, in this screen shot the Principle Tree custom property editor is invoked. The prototype developer uses this property editor to enter the data for the component instance **Principle Hierarchy**, highlighted in the lower left-hand side of the screen.



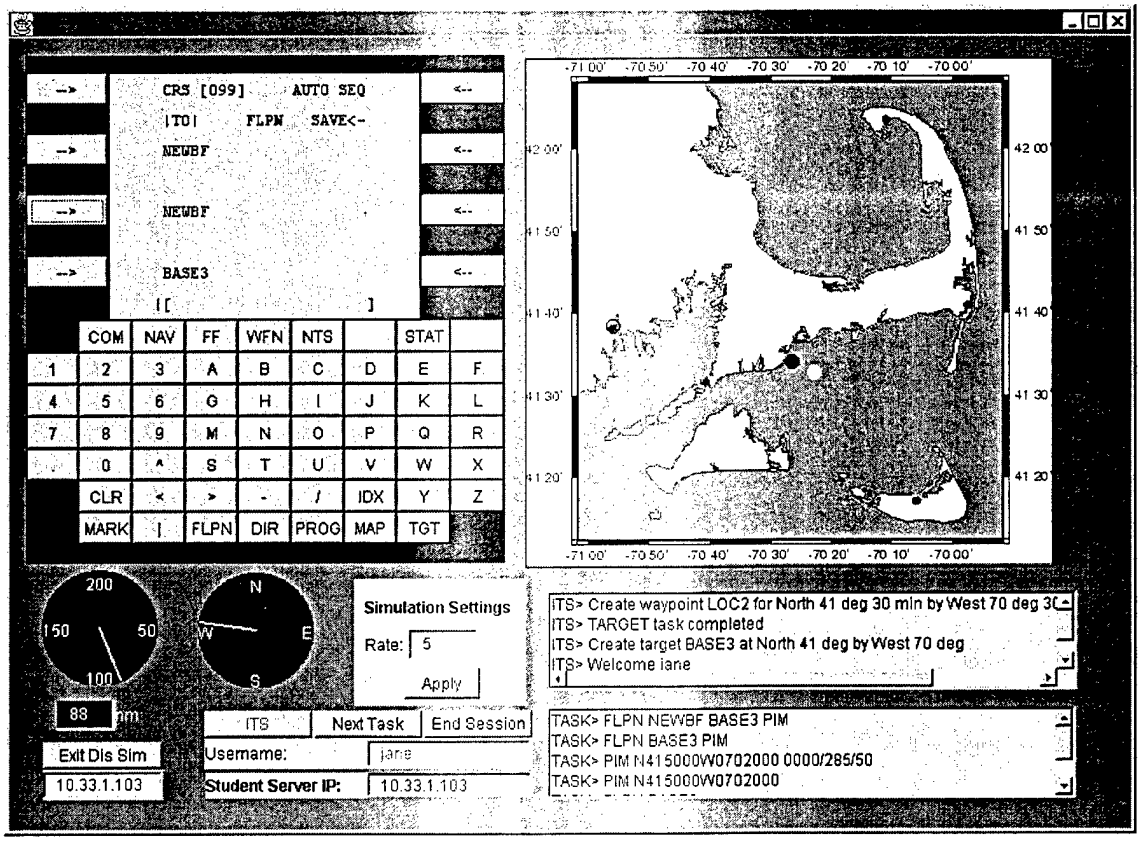
Intelligent Tutoring System Components

This screen shot shows the component instances used in the CDNU part-task trainer to build the Intelligent Tutoring System. Notice how this roughly resembles the component interaction diagram from Figure 8. The ITS component contains references to the Student Database Client component, the Principle Hierarchy component, a Scenario Database component, and the Simulation. The Simulation component is connected to a Distributed Simulation Client. The lower half of the diagram shows the scenario database. This scenario database is defined graphically through a network of tasks and sub-tasks, as described in Section 3.4.



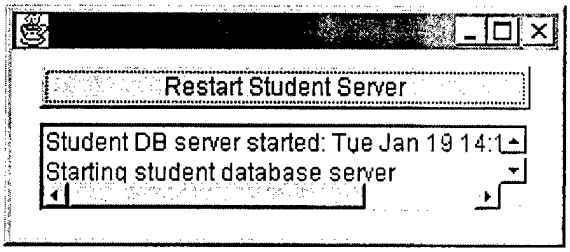
AH-1W CDNU Navigation Part-Task Trainer

This screen shot shows the proof-of-concept AH-1W part-task trainer which was developed during this Phase I. The top-left corner of the screen shows a graphical mock up of the CDNU. This CDNU models all of the navigational functions we wish to train. The map on the top-right corner of the screen provides feedback to the student. The black circle represents the student's helicopter and the gray circles represent other student helicopters within the distributed simulation. Other circles resemble other objects such as PIMs or the current destination. The controls in the lower-left corner are for adjusting simulation settings and specifying the addresses of the simulation and student database servers. The boxes and lower right corner present messages to the user. The bottom one displays every task performed by the student within the simulation. The top one displays messages from the ITS. During this session the ITS asks the student to "Create a Target Base3 at location North 41 by West 70". The student correctly performs this task and the ITS responds with "Target task completed" and then asks the student to create a Waypoint position.



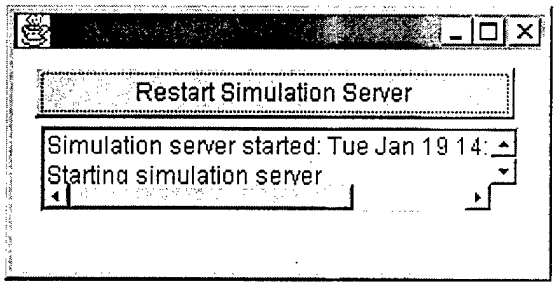
Student Database Server

This screen shot shows the student database server. This server can be run on any machine on the network. The ITS of the Part-Task Trainer uses the JavaRMI distributed object protocol to connect to this server and access and update student model information. To enable an ITS to make use of a remote server all the prototype developer needs to do is create an instance of the Student Database Client component and connect it the ITS component, as demonstrated in the screen shot titled *Intelligent Tutoring System Components*.



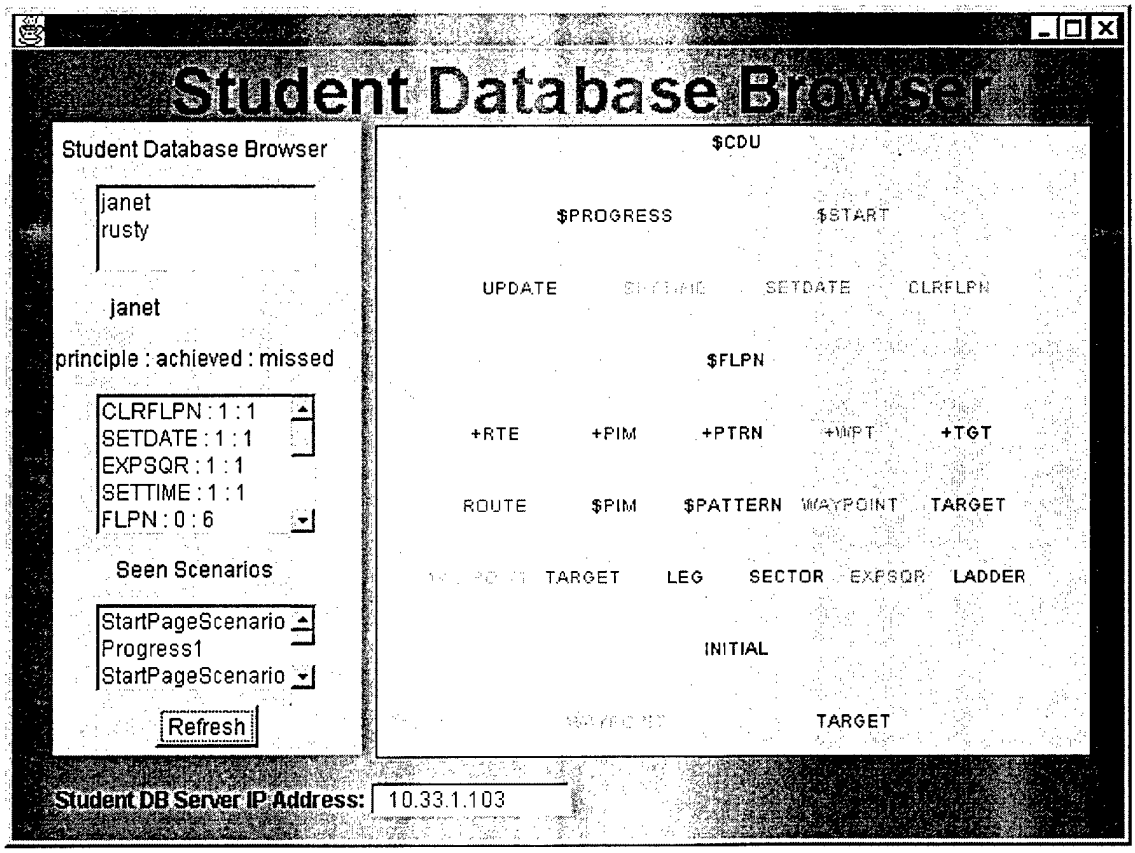
Simulation Server

This screen shot is the Simulation Server. Like the Student Database Server, this server may run on any machine on the network. Students enter the distributed simulation by pressing the "Enter Dis Sim" button in the lower-left corner of the part-task trainer. This server then makes sure every part-task trainer in the simulation remains in sync. For this proof-of-concept prototype, this only means that everyone within the distributed simulation can see one another on their map displays. However, one can imagine how this could be used for more advanced applications, such as team coordination training or linking several different types of part-task trainers together.



Instructor Utility

This screen shot shows an instructor utility which was developed during the Phase I. This utility is used by instructors to monitor the students' progress. This instructor utility could be built in a very short amount of time, because it reuses many of the software components from the Part-Task Trainer prototype. The instructor may select a student from the box in the upper-left corner. The browser will then display all the principles the student has achieved and missed as well as a list of scenarios that the student has been exposed to. The graph on the left is color coded to graphically illustrate the students understanding of the principles. Red resembles principles the student has missed every time, green represents principles the student has achieved, and blue represents the principles the student has not yet been presented. This is a relatively simple application. One can imagine how it might be extended such that an instructor could fine-tune the student model or request a particular student be presented a particular scenario, for example. These extensions would be very straight-forward to add.



SHAI *Stottler Henke Associates, Inc.*
1660 S. Amphlett Blvd., Ste. 350
San Mateo, CA 94402
(650) 655-7242
(650) 655-7243 (FAX)
http://www.shai.com

Certification of Technical Data Conformity (May 1987)

The Contractor, Stottler Henke Associates, Inc., hereby certifies that, to the best of its knowledge and belief, the technical data delivered herewith under Contract No. N68335-98-C-0147 is complete, accurate, and complies with all requirements of the contract.

Wayne R. King Jr.
Signature

Wayne R. King, Jr.
Name - printed

Controller
Title

January 20, 1999
Date

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 20, 1999	Final Progress Report

4. TITLE AND SUBTITLE
Semi-Automated Part-Task Trainer Prototype Development Environment

5. FUNDING NUMBERS
N68335-98-C-0147

6. AUTHORS
Rusty Kinnicut and Richard Stottler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Stottler Henke Associates, Inc.
1660 South Amphlett Blvd., Suite 350
San Mateo, CA 94402

8. PERFORMING ORGANIZATION REPORT NUMBER
Report #172: Trainer-PDE Final Report

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Air Systems Command HQ - Attn: William Walker, PMA2052C
Bldg. 2272 - Suite 345
47123 Buse Road Unit IPT
Patuxent River, MD 20670-1547

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
NONE

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In this Phase I SBIR research, we proved the feasibility of a semi-automated part-task trainer (PTT) prototype development environment. In this effort, we set out to address the shortfalls of traditional PTTs and to introduce new concepts into the PTT prototyping process. The research developed innovative rapid prototyping techniques for the development of PTT prototypes. These techniques are leveraged upon the state-of-the-art software engineering methodologies of component oriented programming (COP), distributed objects, and visual development environments. This means our environment will maximize software reuse and support a number of COTS tools. This innovation will result in a dramatically reduced cost of PTT prototype development and maintenance. Further, prototypes built with this system will employ intelligent tutoring systems (ITSs) which tailor training programs based on detailed mental models of the students. Techniques for defining ITSs using the COP paradigm were designed during this project. Finally, the prototyping environment enables the development of PTTs which take full advantage of intranets and the Internet. This enables features such as distributed simulations, team training, and student monitoring utilities to be integrated into PTT prototypes. A proof-of-concept prototype was developed to demonstrate the key concepts and a full operational system was designed.

14. SUBJECT TERMS
Part-Task Trainer (PTT) Intelligent Tutoring System (ITS) Artificial Intelligence (AI)
Component Oriented Programming (COP) Simulation Environment Rapid Prototyping

15. NUMBER OF PAGES
38

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

NSN 7540-01-280-5500 Computer Generated STANDARD FORM 298
(Rev 2-89)