ED 362 020 FL 021 473

AUTHOR	Tomlin, Russell S.; Douglas, Sarah A.
TITLE	Beginning Second Language Instruction. Computer-Based Curriculum Improvements.
INSTITUTION	Oregon Univ., Eugene. American English Inst.
PUB DATE	89
CONTRACT	G008541129
NOTE	38p.
PUB TYPE	Reports - Descriptive (141) -- Reports - Evaluative/Feasibility (142)
EDRS PRICE	MF01/PC02 Plus Postage.
DESCRIPTORS	Authoring Aids (Programing); Communicative Competence (Languages); *Computer Assisted Instruction; Computer Simulation; *Computer Software; Curriculum Development; Higher Education; *Listening Comprehension; Oral Language; *Second Language Instruction; Second Language Learning; Undergraduate Students
IDENTIFIERS	Project Teach (FIPSE); University of Oregon

ABSTRACT

This project developed computer-based language teaching software to assist beginning second language learners develop listening comprehension skills. Students interact with computer-based simulations of real world problems, which require their understanding of oral language provided by the computer system. The project embraced a communicative approach to language teaching and learning that placed language learning in a task-oriented, problem-solving, communicative context. It also embraced programming by rehearsal in which teachers utilize a theatrical metaphor (teacher as director) to develop novel simulations within the object-oriented authoring component of the computer tutor system, LingWorlds. The project's target population was 2100 undergraduate students beginning foreign language study in several language departments (Romance, Germanic, Russian, Japanese, Mandarin, and the American Language Institute) at the University of Oregon. LingWorlds is detailed in the project description. The end result of the project is the creation of an intelligent computer-assisted instruction system. (Contains 48 references.) (Author/JP)

Beginning Second Language Instruction
Computer-Based Curriculum Improvements

Cover Sheet

ED 362 020

Grantee Organization:

University of Oregon
American English Institute
241 PLC.rt
Eugene, OR 97403

Grant No.:

G008541129

Project Dates:

Starting Date: September 1, 1985
Ending Date: August 31, 1988
Number of Months: 36

Project Directors:

Russell S. Tomlin
Department of Linguistics
University of Oregon
Eugene, OR 97403
(503) 686-3909
tomlin@fog.uoregon.edu

Sarah A. Douglas
Department of Computer & Information Sciences
University of Oregon
Eugene, OR 97403
(503) 686-3974
douglas@cs.uoregon.edu

Fund Program Officer:

Sandra Newkirk

Grant Award:

Year 1:	**$87,235**
Year 2:	**$86,457**
Year 3:	**$52,780**
Total:	**$226,472**

FL021473

Summary

This project developed computer-based language teaching software to assist beginning second language learners develop listening comprehension skills. Students interact with computer-based simulations of real world problems, which require their understanding of oral language provided by the computer system. The project combined two independent insights. First, it embraced an approach to language learning and teaching called the *communicative approach* (along with a more narrowly defined sibling, the *comprehension approach*), which emphasize placing language learning in a task-oriented, problem-solving, communicative context. Second, it embraced an approach to computer tutor design called "programming by rehearsal", in which teachers utilize a theatrical metaphor (teacher as director) to develop novel simulations within the object-oriented authoring component of the computer tutor system. The project developed software to produce and operate such interactive language learning simulations and associated materials to assist language teachers in developing communicatively oriented simulations.

Beginning Second Language Instruction

Computer-Based Curriculum Improvements

FIPSE Grant: G008541129

Russell S. Tomlin
Sarah A. Douglas
University of Oregon

Executive Summary

A. Project Overview

This FIPSE project was directed at improvements in undergraduate foreign language education at the University of Oregon (and similar institutions across the U.S.). More specifically, the project was directed at improving foreign language instruction for beginning level students and in the specific area of listening comprehension. At project conception, we believed that a new kind of computer-based support system could be created which would permit students to engage in listening oriented communicative interactions with simulations of real world problems.

The principal outcomes of the project were: (1) user friendly software for use by teachers in designing and by students in interacting with language teaching simulations, and (2) new insight into the nature of language learning and teaching. Both of these should have some impact on foreign language instruction and on the design of other software systems.

B. Purpose

Oral communication skills—the ability to comprehend and produce oral discourse—are crucial in nearly every educational, business, and scientific setting of language use. Yet the development of oral communication skills remains a difficult theoretical and practical problem, and traditional language teaching approaches regularly fail to help many learners. The central problem addressed by this project was to design a computer assisted language instruction system which could help beginning language learners develop their aural comprehension abilities. The reasons for targeting beginners as well as listening comprehension were three. First, relatively little software is directed at true beginners. Second, relatively little software is directed at improving listening comprehension. Third, the computer environment is best suited to work in teaching listening.

C. Background and Origins

Origins

Given an interest among our graduate students in computer-assisted language instruction and our examination of available systems, we believed existing software and the approaches to language teaching taken under them to be generally unsatisfactory, both pedagogically and computationally. From a pedagogical viewpoint, existing CALL (computer assisted language learning) software did not incorporate innovations in second language teaching approaches otherwise important to the field, in particular task-oriented, problem-solving approaches subsumed under the general *communicative approach*. From a computational viewpoint, existing CALL utilized very simplistic programming strategies and control structures while not permitting easily truly creative teacher initiated variation in student lessons.

Institutional Context

The narrow target of our project was undergraduate students beginning foreign language study at the University of Oregon, a population of some 2100 students. These students were distributed in several languages departments: Romance, Germanic, Russian, East Asian (Japanese and Mandarin), and the AEI.

In addition, we had resources available to us through the American English Institute (AEI). The AEI represents a rather unique context for this FIPSE project. Unlike most English language teaching units on major university campuses, the AEI, in its operational charter specifies that it must support and promulgate research in second language learning and teaching. Thus, in addition to the direct grant support by FIPSE, the AEI provided substantial material support to the project in the form of new equipment and additional personnel support.

D. Project Description

Overall, the principal goal of this project was to develop a second language teaching software system to assist nil proficiency learners develop beginning listening comprehension skills. We entered the project with a number of elaborated examples of the kind of system we wanted to produce, but without great certainty about the precise path that would get us there. In some respects, one of the most important outcomes of the project, besides the actual software developed, was the development of understanding of what some of the basic issues are in language learning and teaching that must be faced in software development as well as what some of the basic issues are in software design and implementation. As we proceeded, and even as we continue now, we redefined our project goals to pursue some fundamental questions in this learning/teaching domain that conventional wisdom did not address.

Language Learning and Language Teaching

From the point of view of language teaching theory our project draws on two important and innovative approaches to second language learning and teaching: the communicative and the comprehension approaches. Proponents of the communicative approach argue that successful language learning occurs when the student is provided the opportunity to solve non-language problems using the developing second language (Widdowson, 1978; Krashen and Terreli, 1983). They criticize traditional language teaching for focusing too much effort on the conscious discussion and manipulation of rules of language usage and not enough effort on the acquisition of the second language grammar through efforts to use that grammar to solve actual communication problems. This philosophy integrates well with the general spirit of ICAI wherein learning is a problem-solving process.

Proponents of the comprehension approach argue that second language learning is enhanced when beginning stages of language learning are devoted to developing the ability to understand the second language. Obligatory oral production is delayed until the student is able to understand easily utterances in the second language. Delaying production improves student performance in other aspects of language acquisition (Postovsky, 1974, 1976; Asher, 1966, 1969, 1972, 1974, 1981; Winitz, 1981).

Our project embraces both of these complementary approaches to language learning and teaching. The instructional system we have created involves the student in solving communicative problems interactively with the system. The student participates in problem-solving simulations which allow manipulation of objects in a physical scenario or *microworld*. Information about the problem to be solved as well as information about the microworld is given in the second language. Meta-level commentary by the tutor is also in the second language. The teaching intervention in these simulations can vary from highly directed to coaching to purely student-controlled exploration.

Design Approach

Our pedagogical model requires many small problem-solving environments to be built, as well as many expert tutors. This has motivated us to study the possibility of building a high-level authoring system. Our general philosophy in constructing this system is that we want the microworlds to be very knowledge-intensive and totally integrated with the interface. We also want them to be reusable. These two themes have pushed us to envision a sort of library of microworlds and tutoring components. LingWorlds offers the teacher a rather large amount of programming power, if the teacher wants to use it, while permitting teachers with less experience the facility to build simple simulations.

Authoring lessons for computer-based second-language instruction is, traditionally, a time-consuming and intensive task which has all of the problems of interface construction. With the advent of bit-mapped displays and mice, the author is faced with ever increasing design complexity as graphics and sound supplement text displays, windows allow multiple contexts for user tasks, and pointing devices join keyboards. On sophisticated systems such as Interlisp on the Xerox 1100's, the interface programming effort has been informally estimated to consume about 80% of the total programming time. The long delay experienced in software development for even less complex machines such as the Macintosh can similarly be attributed to the complexity of composing over 600 ROM-based interface functions into usable interactive programs.

For later purposes of building the authoring system as well as generalizing microworlds, we are committed to three design methodologies: rapid prototyping, taxonomic classification, and direct manipulation for the interfaces. We feel that these methods provide greater programming productivity through the ability to custom-tailor existing code by specializing subclasses and adding instances, and by immediate simulation of of code modules which shortens the generate-and-test loop. These three approaches are manifested in object-oriented programming environments like Smalltalk and Lisp with Flavors. Previous work comparing rule-based with object-oriented ICAI tutors (Douglas, 1986) suggested that there are significant advantages possible with an object-oriented implementation, not the least of which is the accommodation of event-driven, student-initiated control with event-driven, tutor-initiated control. Thus, our system, like Programming by Rehearsal, ARK, Thinglab, and STEAMER, is almost entirely object-oriented.

AN EXAMPLE: *PROVISIONING THE LIFEBOAT*

In order to provide the reader with a concrete idea of how the system works, the following is an extended example of one microworld called *Provisioning the Lifeboat*. In this simulation the student is faced with the non-linguistic task of provisioning a lifeboat before an ocean liner sinks. The simulation begins with an introductory animation and instructions. The computer displays an animation showing an ocean liner approaching an iceberg. A simple oral narrative accompanies the animation. As the ship collides with the iceberg, a second, close-up animation occurs showing the sinking ship. Finally, an on-deck scene of equipment and people near a lifeboat is presented. At this point in the simulation, student interaction begins. Oral language directs the student to locations of provisions and equipment needed to use the lifeboat successfully. The student responds to the instructions by pointing, clicking or dragging with a mouse. Only through successful comprehension of the spoken language will the student be able to complete the tasks required to use the lifeboat.

While commands can be relatively simple to begin ("Put the anchor in the lifeboat."), they can become increasingly complex ("Put the binoculars in the basket in the boat and then put the waterjug beside them."). Note that in this second problem the student must solve several complicated problems: know the names of the objects, perform the task in the correct temporal order, select which of the two baskets is the correct location, and understand prepositions of location (beside, in, etc.).

There are three versions of the *Lifeboat* simulation which vary the mix of student and tutor initiation and control complexity: an exploratory (student-initiated), a game, and a directed tutor. They demonstrate the kind of versatility that we want to provide for microworld control. In the ***exploratory mode***, the student can use the mouse to single-click objects and hear their names ("The basket.") or double-click objects to hear a linguistic description of their locations in relation to another object ("The woman is not in the lifeboat.") The student is also free to drag objects from one location to another to hear the effect of a changed location. Help instructions are available by clicking a question mark (?) icon. In the ***game mode***, oral commands direct the student to locations of provisions and equipment needed to provision the lifeboat successfully. The sinking of the ship provides time pressure for completion of the tasks. The system keeps a score of successful tasks.

Finally, in the ***tutored lifeboat mode***, the directed tutor mode supplements the oral commands of the game mode with remedial intervention. The control strategy that we use is derived from protocol studies of a human expert tutor. The tutor maintains a curriculum of concepts to be taught and a differential student model to determine state transition information and to diagnose types of errors. If the student fails to move any object after a few seconds, the command is repeated. A second such failure causes repetition of the instructions for the overall task; after the third failure the system demonstrates the action. If the student moves an object to the wrong place, it is returned (by the system) to its initial location and the command is repeated. After several failures, the system will move the object to the appropriate location, demonstrating the task. The student is then given another opportunity.

These examples develop the comprehension-based grammar of the nil-proficiency learner in at least the following ways:

- Development of lexicon/vocabulary.
- Development of the grammar of spatial relations.
- Development of the grammar of reference.
- Development of basic syntax and word order.
- Development of the English article system.

From the point of view of the student, we believe that the system we have developed will be challenging and interesting. The student cannot accomplish the needed tasks simply through knowledge of the world, but must comprehend the second language utterances. That is, the student is engaged in true communicative behavior, using the developing second language to solve meaningful, non-language problems--the essence of the communicative approach. The student demonstrates mastery of the linguistic task by physical manipulation of the world rather than by linguistic production. Thus, the student works on aural comprehension, the essence of the comprehension approach to second language learning.

E. Project Results

Pedagogical Efforts

Complementary to the general problem of developing the computer tutor are a number of pedagogical issues pursued during the project: (1) the nature of tutoring, (2) the nature of language teaching simulations, (3) specific curriculum and simulation lessons to be implemented when the system was ready, and (4) general principles for developing listening abilities at beginning levels.

1. ***The nature of tutoring.*** Perhaps the most fundamental question for this project was posed early on: what is it that tutors teach and how do they decide what to do at any given point in a tutorial interaction? In our efforts,we developed an approach which yielded understanding of what it is tutors do when they teach. Using a limited semantic domain, we videotaped expert language tutors engaged in one-on-one tutorials with nil proficiency learners of various languages. These observations were taken both of face-to-face tutorials and of computer mediated tutorials (tutorials where, like the computer, the tutor did not have access to visual information on the movements of learner face and hands and eyes). From these observations, we have been able to extract significant components and principles of second language tutorials (Douglas, in press; Tomlin, Douglas et. al., 1988).

2. ***The development of model language teaching simulations.*** During the course of the project we developed a number of listening oriented language teaching simulations. They represent the kinds of simulations we believe will prove effective pedagogically and engaging to the language learner.

(1) **The lifeboat.** In this simulation the student must provision a lifeboat in response to oral instructions in a second language (either ESL or Japanese). The student moves objects about the computer screen and places them in requested positions in the lifeboat. In order to do this successfully, the student must comprehend lexical expressions for the various items and relational expressions for the targeted locations.

(2) **Maptiles.** This simulation represents one of our ideal lesson environments. In the development of listening abilities it is quite common to use maps to represent a territory through which the student must navigate in response to input in the target language. The maptiles environment provides a toolbox with which teachers can construct their own maps and language lessons based on them.

(3) **The animated dictionary.** The animated dictionary (AD) is a lexical support system for students and teachers. Students use the animated dictionary to search for lexical items, related items and collocations, a variations from semantic prototypes. Teachers control the extent of student access in a given scenario.

(4) **FlatLand.** In this simulation, the student builds simple two-dimensional configurations from a limited set of objects: black or white, large or small, circles or squares.

(5) **Mystery world.** This simulation places the student in the position of a witness to some set of events. The events are presented as an animated film of some kind or other. Associated with the film is a descriptive sound track which relates the actions witnessed by the student.

3. ***The nature of simulations.*** In order to assist teacher in the design of engaging and effective language teaching simulations, we examined the underlying organization of task-oriented/language teaching simulations and developed a taxonomy of simulation problem types.

4. ***General principles of communicative language teaching targeting listening comprehension.*** We effected a search of the pedagogical and theoretical literature in second language learning and teaching and we interviewed practicing second language teachers in order to create an inventory of listening activities. The inventory of activities is organized to reveal: (1) the necessary prerequisite skills required to perform the task, (2) the learning goals for the task, (3) the expected outcomes for the task, (4) the means/methods of presenting/executing the task, and (5) means of evaluating the effectiveness of the task.

The LingWorlds Simulation System: Implementation

The LingWorlds tutoring system consists of two component parts. One component is seen and used by the learner to engage in language learning simulations. This component we will call the **tutorial system**. The second component is used by language teachers to create simulations and language learning problems. We will call this component the **authoring system**.

The LingWorlds Tutorial System

The LingWorlds microworld consists of sequences of scenes of objects. Each object has a graphic display and is capable of performing various programmable actions such as speaking and moving in response to 1) student interactions such as clicking or dragging the object, 2) actions directed by other objects and 3) internally generated tutor actions. The language generator contains special routines which manipulate digitized data and are written in assembly language to be as efficient as possible. The language generator has a digitized lexicon, and a case-frame semantic grammar, and is capable of dynamically generating utterances from a conceptual representation. It is much too primitive to be considered a full-fledged natural language generator but is sufficient for our simple language teaching. A separate generator is needed for each language taught.

LingWorlds microworlds are generated by the authoring system (see below). Each object is essentially an object-oriented programming construct, as is the tutor component. If the student clicks with the mouse on a particular object on the screen, a method is appropriately activated which may move the object, cause it to say something, etc. This is the basic flavor of all exploratory learning environments. It is the case, though, that we often want to introduce more teaching intervention into student actions. Control from the tutor is introduced into LingWorlds by the tutor object. In LingWorlds the control is knit together by message-passing between objects and the tutor object. A totally exploratory microworld has its control locally defined with the behavior of each object tied to student actions. A game microworld increases tutor control. Finally, a goal-directed tutor microworld controls most of the interaction through a task-based agenda. Even in the goal-directed tutor, objects still retain individual control over the semantics of their own actions. For example, the tutor can be notified that a particular object has been dragged, and through additional message-passing with the object determine the location. Thus, the object-oriented paradigm allows an event-driven format that accommodates user-initiated actions as well as internally controlled actions.

The LingWorlds Authoring System

Briefly, the goals of the authoring system were to enable a non-programming teacher of language to build microworld-based lessons. This is achieved by three types of novice programming: direct manipulation, menu-based selection and a very easy to learn programming language. Using the authoring system, functioning lessons can be developed, tested, and refined very quickly. The authoring system allows immediate switching between creating the microworld and testing it.

Direct manipulation is used to position objects in a scene and trace animation paths. Menu-based selection is used for choosing graphic images, selecting attributes of objects (such as whether they are draggable), and choosing system functions such as testing a microworld. The programming language is composed of English like actions and is used to create functionality for objects. Objects have a (growing) set of primitive actions, such as *flash*, *highlight*, *say*, and *move*. There are also control primitives such as *if* and *repeat while*. From these primitive actions and controls, users can create new actions. Taken together, the built-in primitive and user-composed actions are available to all things and are called "global actions." Each thing also has a set of six "local actions," which are functional responses of the specific thing to student events. The local actions are *1 click action*, *2 click action*, *dragging action*, *landing action*, *evaluation action*, and *history action*. The actions, both global and local, are written by the teacher in a semantically based structure editor using hierarchical dialogues. There is no typing of text, simply selection of pop-up menu items. The system makes sure that the menu items presented to the user are syntactically and semantically appropriate.

The authoring system produces sound by dynamically generating speech from a digitized phrasal lexicon. The sounds are recorded using SoundCap™ or SoundWave™, and then converted into resources which can be selected as arguments to the *speak* method. Thus, for example, one might use three separate sounds, "the lantern," "is in," and "the lifeboat," which are suitably inflected to produce the unified coherent utterance "The lantern is in the lifeboat." Sound generation can be context-dependent since an object can have a state history. Thus different utterances for an object can be generated in response to different situations.

The LingWorlds authoring system is an interpreter implemented on the Macintosh II computer in Allegro Common Lisp using Allegro's Object Lisp system. It generates Lisp code which can be supplemented by additional Lisp programming if so desired. Technically, the system is built from a set of primitive features, which are either constructed in the authoring system (such as text) or are imported from other programs (such as SoundWave™ and MacPaint™). These primitives include digitized sound, Macintosh-style images, locations, integers, booleans, text, and icons.

Evaluation

Our informal evaluation of the system indicates that the average time to prototype a microworld is about several orders of magnitude less the time taken by Pascal programmers on the Mac (days rather than months). We have yet to extensively test the system with more formal evaluation studies, but those are planned for the future. Since the system is actually quite a powerful system for creating *any* instructional software, a version of the system was recently requested and sent to Yale University for use in building instructional software for mathematics tutoring.

We have built both an English and Japanese version of *Provisioning the Lifeboat*, with all the expected savings in programming. We built the exploratory version first, and then added the directed tutor version. A game version has not been programmed, but would be trivial. Since interactors and tutors can be specialized for any particular microworld. most of the code is inherited and reusable. The Japanese version of the *Lifeboat* is virtually identical to the English except for the addition of animacy features required for speech synthesis. Instantiation allows easy copies of objects with the minimum of programming effort. We have also built a microworld called *Flatland*, in which the student is taught geometric shapes, colors, size and spatial relations. This is a directed tutor version, based on extensive protocols with human tutors. The implementation of *Flatland* took one day, since almost all of the code was reusable from the *Lifeboat* problem.

F. Summary and Conclusions

In this summary we have attempted to describe the principal outcomes and problems associated with our project to build an interesting and effective computer-based second language tutor, LingWorlds. Over the course of the project, we found that the basic problems of interest to us in creating this system increased in complexity and scope. However, the end result of the project is the creation of a ICAI system that teachers will be able to use coupled with important new insights into the nature of language learning and teaching.

Final Report

A. Project Overview

This FIPSE project was directed at improvements in undergraduate foreign language education at the University of Oregon (and similar institutions across the U.S.). More specifically, the project was directed at improving foreign language instruction at the beginning level and in the specific area of listening comprehension. At project conception, we believed that a new kind of computer-based support system could be created which would permit students to engage in listening oriented communicative interactions with simulations of real world problems.

The principal outcomes of the project were: (1) software for use by teachers in designing and by students in interacting with language teaching simulations, and (2) new insight into the nature of language learning and teaching. Both of these have impact on foreign language instruction and on the design of other instructional software systems.

B. Purpose

Oral communication skills—the ability to comprehend and produce oral discourse—are crucial in nearly every educational, business, and scientific setting of language use. Yet the development of oral communication skills remains a difficult theoretical and practical problem, and traditional language teaching approaches regularly fail to help many learners. The central problem addressed by this project was to design a computer assisted language instruction system which could help beginning language learners develop their aural comprehension abilities. The reasons for targeting beginners as well as listening comprehension were three. First, relatively little software is directed at true beginners. Second, relatively little software is directed at improving listening comprehension. Third, the computer environment is best suited to work in teaching listening.

From the point of view of language teaching theory our project draws on two important and innovative approaches to second language learning and teaching: the communicative and the comprehension approaches. Proponents of the communicative approach argue that successful language learning occurs when the student is provided the opportunity to solve non-language problems using the developing second language (Widdowson, 1978; Krashen and Terrell, 1983). They criticize traditional language teaching for focusing too much effort on the conscious discussion and manipulation of rules of language usage and not enough effort on the acquisition of the second language grammar through efforts to use that grammar to solve actual communication problems. This philosophy integrates well with the general spirit of ICAI wherein learning is a problem-solving process.

Proponents of the comprehension approach argue that second language learning is enhanced when beginning stages of language learning are devoted to developing the ability to understand the second language. Obligatory oral production is delayed until the student is able to understand easily utterances in the second language. Delaying production may even improve student performance in other aspects of language acquisition (Postovsky, 1977, 1979; Asher, 1966, 1969, 1977; Winitz, 1981; Winitz & Reeds, 1973).

Our project embraces both of these complementary approaches to language learning and teaching. The instructional system we have created, called Lingworlds, involves the student in solving communicative problems interactively with the system. The student participates in problem-solving simulations which allow manipulation of objects in a physical scenario or *microworld* Information about the problem to be solved as well as information about the microworld is given in the second language. Meta-level commentary by the tutor is also in the second language. The teaching intervention in these simulations can vary from highly directed to coaching to purely student-controlled exploration.

Our pedagogical model requires many small problem-solving environments to be built, as well as many expert tutors. This motivated us to create a high-level authoring system. Our general philosophy in constructing this system is that we want the microworlds to be very knowledge-intensive and totally integrated with the interface. We also want them to be reusable. These two themes have pushed us to envision a sort of library of microworlds and tutoring components. LingWorlds offers the teacher a rather large amount of programming power, if the teacher wants to use it, while permitting teachers with less experience the facility to build simple simulations.

C. Background and Origins

Origins

The historical origins for this FIPSE project involved the interaction and integration of ideas from four distinct disciplines: linguistics, applied linguistics, artificial intelligence, and computer science. Initially, graduate students in applied linguistics, in preparation to be English language instructors, were very much interested in the possibilities represented by computer assisted instruction (CAI or CALL, as it is referred to). However, the PI's, each in his/her own domain, found existing software and the approaches to language teaching taken under them to be unsatisfactory, both pedagogically and computationally. From a pedagogical viewpoint, existing CALL (computer assisted language learning) software did not incorporate innovations in second language teaching approaches otherwise important to the field, in particular task-oriented, problem-solving approaches subsumed under the general *communicative approach*. From a computational viewpoint, existing CALL utilized very simplistic programming strategies and control structures while not permitting easily truly creative teacher initiated variation in student lessons. Simply put, we found our students interested in the possibilities represented by CALL but we felt unable to recommend much of anything to them to study or emulate.

Subsequently, we began to talk about how an interesting CALL system might be made, and we played around with the idea of replicating aspects of Asher's (1977) model of 'total physical response' training within a CALL environment. A student of Tomlin's (Yuri Saul) had mocked up a very simplistic vocabulary learning example on an Apple IIe which showed that the idea had possibilities. Review of this and further considerations by PI Douglas led to the belief that an interesting CALL system could be developed using ideas from knowledge-based (artificial intelligence) and microworlds programming.

The past 15 years have seen reasonable progress in delivering ICAI (Intelligent Computer-Assisted Instruction) systems which can be used in real pedagogical situations (Anderson, 1985; Clancey, 1982; Sleeman, 1982; Soloway et al., 1981). ICAI systems can be contrasted with traditional CAI (Computer-Assisted Instruction) in the following ways:

- The problem of learning and, consequently, teaching is seen as a cognitive, knowledge-intensive, and typically problem-solving process rather than a reinforcement process.
- The control structure is dynamically generated by the interaction of curriculum, student response, and heuristics for diagnosis and tutoring rather than simply stored by the program.
- The domain knowledge being taught is explicitly available for pedagogical decisions rather than embedded as numerical calculations (simulations or drill and practice) or blocks of text/images/sound (programmed instruction).
- Empirical, scientifically conducted studies of students and teachers form the foundation for the research, from initial data collection through to evaluation.

Influenced heavily by the cognitive science movement, ICAI systems represent a significant attempt to model the cognitive aspects of teaching and learning, particularly in providing an individualized approach.

The notion of microworlds was first proposed by Papert (1980) for LOGO programming. A later paper by Burton and Brown (1982) on reactive learning environments urged the pedagogical value of exploratory learning in domains other than programming. Our work has been greatly influenced by research on embedded semantics in microworlds with direct manipulation interfaces: Programming by Rehearsal (Gould & Finzer, 1984), ARK (Smith, 1986), the latest work on STEAMER (Hollan et al., 1986), and Thinglab. However, we intend to build microworlds which are a simulation of the world as represented for tutoring purposes. This places our work closer to the work of Programming by Rehearsal than systems like ARK, STEAMER, and Thinglab which represent the knowledge of

Newtonian mechanics and hydraulics. We are interested in packing as much knowledge as possible into the microworld. Thus we want to allow the system to derive inferences beyond the facts explicitly declared. For example, we want the system to be able to compute spatial relations dynamically. We have found that AI work on scene analysis and diagram understanding, as well as the literature on spatial reasoning and data bases is related to our work. Additionally, while we wish to continue the basic notion of exploratory learning, we also want to introduce more tutor-controlled strategies and ICAI computational mechanisms into the microworlds.

For later purposes of generalizing microworlds as well as building the authoring system, we were committed to three design methodologies: rapid prototyping, taxonomic classification, and direct manipulation for the interfaces. We feel that these methods provide greater programming productivity through the ability to custom-tailor existing code by specializing subclasses and adding instances, and by immediate simulation of of code modules which shortens the generate-and-test loop. These three approaches are manifested in object-oriented programming environments like Smalltalk and Lisp with Flavors. Previous work comparing rule-based with object-oriented ICAI tutors (Douglas, 1986a) suggested that there are significant advantages possible with an object-oriented implementation, not the least of which is the accommodation of event-driven, student-initiated control with event-driven, tutor-initiated control. Thus, our system, like Programming by Rehearsal, ARK, Thinglab, and STEAMER, is almost entirely object-oriented.

At this point, the original FIPSE pre-proposal was written, in some ways as a preliminary test of whether the ideas we were entertaining would be of interest to others. FIPSE seemed an interesting funding source because it was risk and innovation oriented and because it was pragmatically oriented. Both Douglas and Tomlin had substantial commitments to pragmatically oriented work. Tomlin was serving as Director of the University of Oregon's American English Institute (AEI), which provides English language training (ELT) to prospective and matriculated foreign students at the UO. Douglas had served as Director of an academic computing center. Both enterprises put us in touch with practical applications of research. It was at this stage that we began to reformulate our issues so that they would more directly serve educational purposes.

Institutional context

The narrow target of our project was undergraduate students beginning foreign language study at the UO, a population of some 2100 students. These students were distributed in several languages departments: Romance, Germanic, Russian, East Asian (Japanese and Mandarin), and the AEI. We did not cultivate any preliminary relations with these departments (aside from the AEI), though we expected all to become interested as the project developed (which has in fact happened, except for the every small Germanic department). Thus, we knew who the targeted population of terminal users were, but more might have been done to engage language teaching faculty at an earlier stage.

On the other hand, we did have substantial resources available to us through the American English Institute (AEI), which PI Tomlin directed. The AEI represents a rather unique context for this FIPSE project. Unlike most English language teaching units on major university campuses, the AEI, in its operational charter specifies that it must support and promulgate research in second language learning and teaching. In addition, as a self-support unit in the UO College of Arts and Sciences, the AEI is able to direct material support to a project like this. Thus, in addition to the direct grant support by FIPSE, the AEI provided substantial material support to the project in the form of new equipment and additional personnel support.

Perhaps the greatest change in the context over time has been ancillary to the project itself but nonetheless important to its eventual successful articulation at the UO. First, a change in Department head in the huge Romance languages department has resulted in a change in orientation regarding Romance language instruction. In particular, Romance has hired one new faculty member and is searching currently for a second specifically in the area of foreign language instruction. This has added some impetus to greater interaction between Romance and the AEI and Linguistics. Second, the UO has, through efforts by the AEI Director, secured a private gift of some $200,000 to be used specifically for the redesign and rebuilding of UO language lab facilities, including computer equipment. Both of these developments represent improvements in the UO context which will help set the stage for incorporation of our FIPSE project results into the UO foreign language instructional setting.

D. Project Description

Overall, the principal goal of this project was to develop a second language teaching software system to assist nil proficiency learners develop beginning listening comprehension skills. We have built a knowledge-based tutoring system for teaching beginning oral communication skills for second (natural) languages. We entered the project with a number of elaborated examples of the kind of system we wanted to produce, but without great certainty about the precise path that would get us there. In some respects, one of the most important outcomes of the project, besides the actual software developed, was the development of understanding of what some of the basic issues are in language learning and teaching that must be faced in software development as well as what some of the basic issues are in software design and implementation. As we proceeded, and even as we continue now, we redefined our project goals to pursue some fundamental questions in this learning/teaching domain that conventional wisdom did not address.

LingWorlds As a Tutoring System

AN EXAMPLE: *PROVISIONING THE LIFEBOAT*

In order to provide the reader with a concrete idea of how the system works, the following is an extended example called *Provisioning the Lifeboat*. In this simulation the student is faced with the non-linguistic task of provisioning a lifeboat before an ocean liner sinks. The simulation begins with an introductory animation and instructions. The computer displays an animation showing an ocean liner approaching an iceberg. A simple oral narrative accompanies the animation. As the ship collides with the iceberg, a second, close-up animation occurs showing the sinking ship. Finally, an on-deck scene of equipment and people near a lifeboat is presented (Figure 1). At this point in the simulation, student interaction begins. Oral language directs the student to locations of provisions and equipment needed to use the lifeboat successfully. The student responds to the instructions by pointing, clicking or

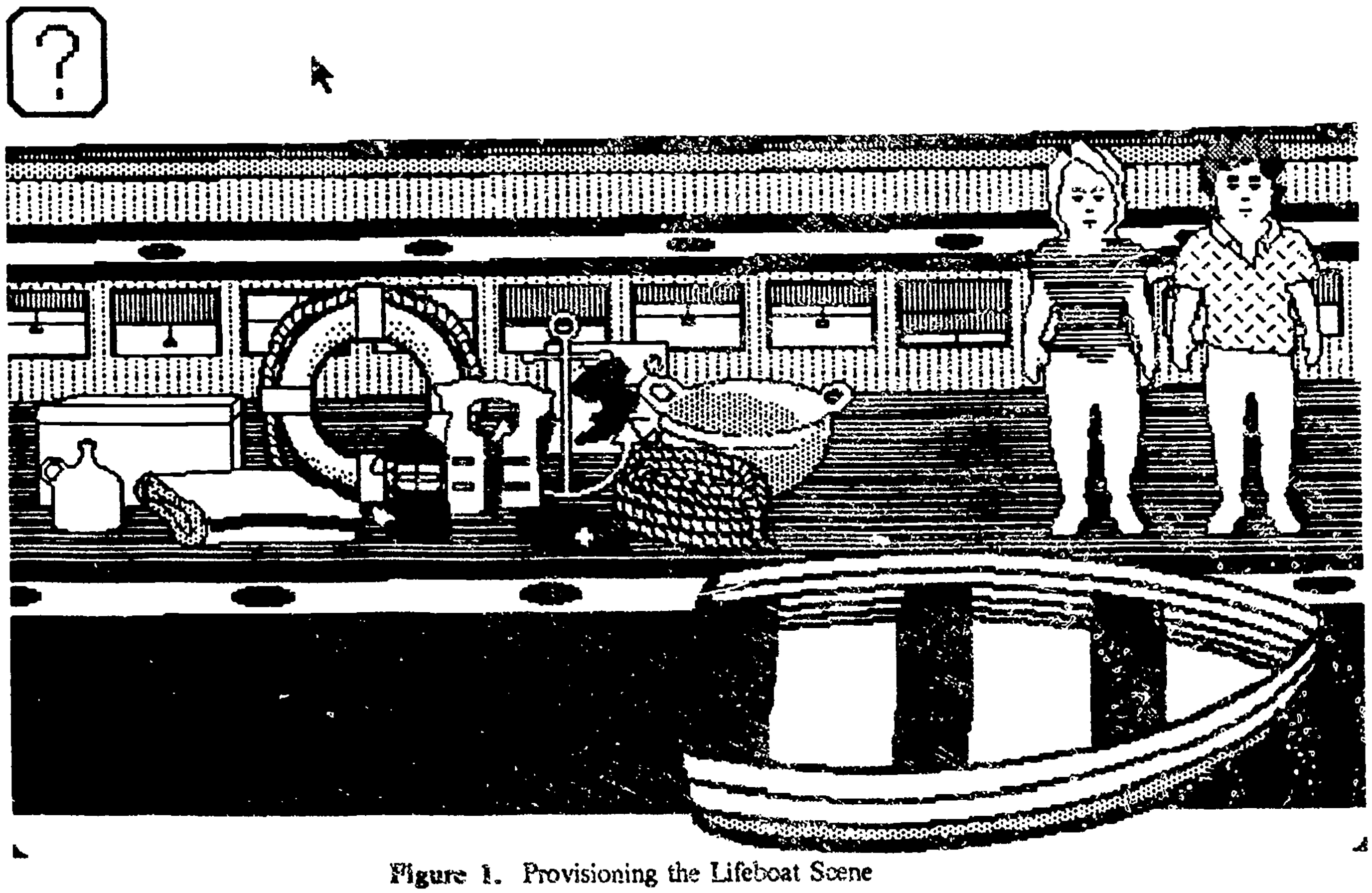


Figure 1. Provisioning the Lifeboat Scene

dragging with a mouse. Only through successful comprehension of the spoken language will the student be able to complete the tasks required to use the lifeboat.

While commands can be relatively simple to begin ("Put the anchor in the lifeboat."), they can become increasingly complex ("Put the binoculars in the basket in the boat and then put the waterjug beside them."). Note that in this second problem the student must solve several complicated problems: know the names of the objects, perform the task in the correct temporal order, select which of the two baskets is the correct location, and understand prepositions of location (*beside*, *in*, etc.).

Three versions of *Lifeboat* are described below which vary by the mix of student and tutor initiation and control complexity: an exploratory (student-initiated), a game, and a directed tutor. This demonstrates the kind of versatility that we want to provide for microworld control.

Exploratory Lifeboat. In a totally exploratory mode, the student can use the mouse to single-click objects and hear their names ("*The basket.*") or double-click objects to hear a linguistic description of their locations in relation to another object ("*The woman is not in the lifeboat.*") The student is also free to drag objects from one location to another to hear the effect of a changed location. Help instructions are available by clicking a question mark (?) icon. (See Figure 1.)

Game Lifeboat. During a game format, oral commands direct the student to locations of provisions and equipment needed to provision the lifeboat successfully. The sinking of the ship provides time pressure for completion of the tasks. The system keeps a score of successful tasks.

Tutored Lifeboat. The directed tutor mode supplements the oral commands of the game mode with remedial intervention. The control strategy that we use is derived from protocol studies of a human expert tutor. The tutor maintains a curriculum of concepts to be taught and a differential student model to determine state transition information and to diagnose types of errors. If the student fails to move any object after a few seconds, the command is repeated. A second such failure causes repetition of the instructions for the overall task; after the third failure the system demonstrates the action. If the student moves an object to the wrong place, it is returned (by the system) to its initial location and the command is repeated. After several failures, the system will move the object to the appropriate location, demonstrating the task. The student is then given another opportunity.

These examples develop the comprehension-based grammar of the nil-proficiency learner in at least the following ways:

- Development of lexicon/vocabulary.
- Development of the grammar of spatial relations.
- Development of the grammar of reference.
- Development of basic syntax and word order.

From the point of view of the student, we believe that the system we have developed will be challenging and interesting. The student cannot accomplish the needed tasks simply through knowledge of the world, but must comprehend the second language utterances. That is, the student is engaged in true communicative behavior, using the developing second language to solve meaningful, non-language problems--the essence of the communicative approach. The student demonstrates mastery of the linguistic task by physical manipulation of the world rather than by linguistic production. Thus, the student works on aural comprehension, the essence of the comprehension approach to second language learning.

E. Project Results

This section describes the LingWorlds simulation system, both in terms of its actual implementation and its associated pedagogical insights.

Pedagogical Efforts

Complementary to the general problem of developing the computer tutor are a number of pedagogical issues we pursued during the project. These efforts fall into five classes: (1) the nature of tutoring, (2) the nature of language teaching simulations, (3) specific curriculum and simulation lessons to be implemented when the system was ready, (4) general principles for developing listening abilities at beginning levels, and (5) support efforts to the computer tutor directly. We will describe briefly the details of the efforts in each area.

1. *The nature of tutoring.* Perhaps the most fundamental question for this project was posed by Douglas early on: what is it that tutors teach and how do they decide what to do at any given point in a tutorial interaction? An answer to this compound question is needed if the computer tutor is to perform in a way similar to human tutors. The existing literature in communicative language teaching provides only the most general guidelines on this, guidelines which are not specific enough to incorporate in any way in the tutor system. Consequently, we have devoted considerable effort to understanding the nature of individual tutoring.

The communicative literature provides only the most general of assertions regarding what is taught and how that teaching is accomplished. The communicative approach (Widdowson, 1978, 1979; Piepho, 1981) views language learning as a cognitive enterprise in which the learner entertains multiple hypotheses regarding the structure and function of target language constituents in natural discourse contexts until sufficient contextualized input is encountered to settle on and automate the learner's closest approximation of the native speaker norms. This process of creative construction of an *interlanguage grammar* (Selinker, 1972) is facilitated when linguistic input is comprehensible to the learner (Krashen 1977, 1982), when it is of sufficient quantity in a variety of discourse contexts, and when the affective environment does not constrain exploration and risk-taking (Krashen 1977, 1982; Schumann 1978).

There are a number of "tenets" of the communicative approach that can elaborate brielfy the general characterization provided above. Under the communicative approach language is viewed as situated social activity , as efforts of discourse production and comprehension, as *communication.*. Thus, in communicative language teaching:

(1) Systematic attention is paid to functional as well as structural aspects of language (Littlewood 1981:1).

(2) Classroom work is aimed at the situational and contextualized use of language (Piepho 1981:20-21).

(3) Teaching and learning are made observable and transparent through content which is made real to the learner through pictures, sketches, diagrams, and other representations (Piepho 1981:20-21).

(4) Attention is focused on the ability to understand and convey information; i.e. on information transfer (Johnson 1982:163-175).

(5) The learner is seen a responsible partner in learning rather than as an object to be manipulated (Piepho 1981:11-12).

Language teaching represents the effort by the tutor to set up the conditions for learning described above. That is, with more or less finely grained teaching efforts, the tutor seeks to provide to the learner a sufficient quantity of comprehensible input drawn from a wide variety of genuine or authentic discourse contexts (Widdowson 1978, Krashen & Terrell 1983) in an affectively "supportive" environment. While the observations above represent some of the general principles defining the communicative approach, these general principles do little to tell us exactly what teachers manipulate in tutoring and when and how they do it.

In our efforts, then, we developed an approach which yielded understanding of what it is tutors do when they teach. Using a limited semantic domain, we videotaped expert language tutors engaged in one-on-one tutorials with nil proficiency learners of various languages. These observations were taken both of face-to-face tutorials and of computer mediated tutorials (tutorials where, like the computer, the tutor did not have access to visual information on the movements of learner face and hands and eyes). From these observations, we have been able to extract significant components and principles of second language tutorials (Douglas, in press; Tomlin, Douglas et. al., 1989).

(1) Tutorials are organized around a limited set of **rhetorical acts**. A rhetorical act represents a basic, composite unit of tutorial activity. The rhetorical act represents a basic linguistic action taken by an individual in a given discourse context. It represents an attempt by the speaker to direct some action on the part of the listener. A rhetorical act consists of three criterial components:

- An *intentional construct*, or simply intention,
- *Behavioral content,*
- *Preparatory conditions.*

The **intention** of a rhetorical act represents the underlying motive precipitating performance of the act. It is, following Brandt (1984), a mental event, the immediate and proximate cause of particular actions engaged in by the tutor. It is also the principal defining characteristic of particular rhetorical acts.

The **behavioral content** represents the set of actions, mental or physical, which the tutor carries out due to the intention. The behavioral content of a rhetorical act includes a description of the generally desired outcome (goals) of the rhetorical act and of the means of achieving this outcome (methods). For the rhetorical act DESCRIBE OBJECT, the behavioral content includes:

- Goal_1: Tutor directs learner attention to some part of the environment.
- Method_1: Tutor points to object.
- Goal_2: Learner links attended object to linguistic input.
- Method_2: Tutor utters object-name after attention is allocated to object by learner.

Preparatory conditions represent tutor assumptions regarding preliminary or ongoing states of affairs in the tutor, in the learner, or in the world which must be present in order for a given rhetorical act to be executed. For the LingWorlds tutor, relevant preparatory conditions for many rhetorical acts include assumptions regarding:

- state of the learner,
- state of the curriculum,
- comparison of (1) with (2),
- state of the environment (world).

For the act of DESC OBJ, the preparatory conditions include:

- tutor assumes object is available to the learner,
- tutor assumes learner is familiar with the concept of the object.

A rough model of a rhetorical act, then, similar to the informal action model presented in Brandt (1984:128, is presented in Figure 2. Our descriptive efforts reveal that beginning second language tutorials (within the domain we examined) are composed of a limited inventory of these kinds of acts. Thus, the building blocks of the computer tutor should be those rhetorical acts utilized in particular discourse domains.

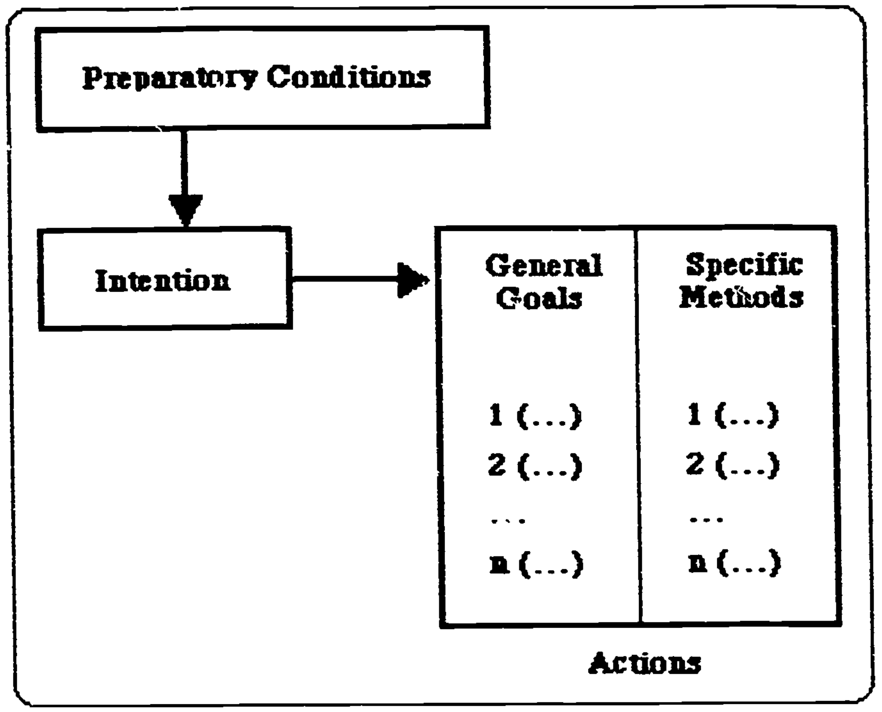


Figure 2. Components of a rhetorical act

(2) ***Tutor evaluation tactics.*** In order to decide what to do next, the tutor must evaluate current actions of the student. In our observations, the information used by the tutor to do this varies according to the nature of the tutorial interaction. In both face-to-face and computer-mediated tutorials, it seems very clear that the tutor makes decisions about which act to select next prior to seeing the actual completion of a student action. That is, if the tutor in order to test some hypothesis like " the student knows with certainty that '*atas* means above' " asks the student to manipulate (in the target language Indonesian) a specific object with respect to another (Put the black square above the small white circle), the tutor will generally decide whether the student has gotten it prior to the action being completed.

In both face-to-face and computer-mediated tutorials, the decision is made on the basis of the extent of uncertainty or hesitation in the student's response. The more hesitant the student is, the more likely the tutor will conclude the student does not yet have control of the relevant part of the grammar. However, the two conditions reveal differences in the tactical information exploited to reach this decision. In the face-to-face tutorials, the decision is made tactically on the basis of information provided by a wide variety of responses: (1) latency in initiating response, (2) hesitation in movement of objects during response, (3) eye gaze, (4) head positions, (5) hand positions. In the computer-mediated tutorials, the visual cues on the student are not available, and the tutor relies primarily on latency and hesitancy in evaluating student performance. Unlike the assumption in virtually all CAI/CALL software we know of, the tutor does not rely solely or even primarily on the correctness of the student response. In the development of the computer tutor then, such observations must be taken into account.

(3) ***The tutor's model of the student.*** Both the human tutor and the computer tutor must maintain a dynamic model of the student and what he has learned of the second language. In our studies, we embraced the general notion that students engage in "creatively constructing" the second language grammar. That is, on the basis of linguistic input which links utterances in the second language to observable situations in the world, the student formulates (not consciously of course) hypotheses regarding the grammar as used to effect communication. Additional input serves

to aid the student to reject false hypotheses and to increase the certainty with which he holds some hypotheses to be true.

The tutor, then, must organize the tutorial effort to take this into account. Thus, the tutor must represent the student's current knowledge of the target language in terms of possible hypotheses and the strengths to which they are held. And, the tutor's actions (selection and use of rhetorical acts) must be tied to the formulation, rejection, and testing of hypotheses by the student.

(4) ***The role of repetition in language learning and teaching.*** There has been great interest in language learning and teaching in understanding the role of repetition (and more generally correction) in language teaching. Our descriptive studies have revealed a number of interesting hypotheses regarding the nature and use of repetition in language learning. First, we see three distinct types of repetition: (1) repetition of the most recent utterance (REP EXP), repetition of a selected part of an utterance (REP FOC), and repetition of an entire utterance/action complex (REP ACT). While the three types of repetition share the gross structural property of similarity in the utterance token, the three differ interestingly in their function and, hence, use by tutors.

The first type of repetition, REP EXP, seems to occur without much, if any, behavioral input from the learner. Typically, then tutor produces a REP EXP immediately after DESC OBJ and independent of any request for the student to do something. For example, at the beginning of our Indonesian learning protocol, one can observe the following:

(1) An example of REP EXP

00:00		**X on LBS**
00:06		**LBS -->**
00:08		**LBS lower right screen**
00:08		Ko:tak/
00:11	03	Ko:tak/

In this example, at the beginning of the tutorial (second 00:00), the tutor produces the rhetorical act DESC OBJ, placing the cursor is on the large black square (LBS), moving the large black square to the lower right hand corner of the display, and uttering an appropriate description of the targeted object. Immediately after the DESC OBJ is complete, the tutor produces a REP EXP, repeating exactly the previous utterance. There is no expectation on the part of the tutor at this point that the student will do anything, yet the repetition occurs nonetheless. It appears that the actual purpose of the repetition in this instance is to allow the student the benefit of retaining in working memory a good representation of the original token. That is, the REP EXP is a social act keyed into the difficulty a beginning learner has in sustaining an auditory presentation of novel linguistic input. Such a hypothesis is supported by comments of learners after the protocols are collected which indicate how difficult subvocalizing the input was for them (even though they were able to perform the tasks required without great difficulty). REP EXPs are used also more at the beginnings of learning segments than in the middles or ends.

Repetitions in which only a part of the previous utterance is repeated, what we call REP FOC (repetition focus), occur at very different times for apparently very different reasons. REP FOCs occur after the tutor requests the student to manipulate some object and the student reveals latency or hesitation in responding. The REP FOC repeats just the targeted linguistic notion in that request. Fro example, consider the data below:

(2) An example of a REP FOC (repeat focus)

13:30	03	2.34	Ko:tak hi:tam ke:cil (.36) DI:ATAS (.15) kotak hitam besar . /
13:35	05	1.11	DiATAS ?/
13:36			**SBS -->**
13:40			**SBS a LBS**
13:40	05	3.81	Baik . /

The utterance at 13:30 represents a request of the student by the tutor to put the small (kecil), black (hitam) square (kotak) above (di-atas) the large (besar), black square. At this stage, the student has already mastered the lexical and structural aspects of Indonesian required to determine the referents of the noun phrases. It is precisely the the spatial relation of 'above' that is being manipulated in this instance. The student hesitates briefly after the end of the utterance (1.11 seconds after completion of 'besar'), and the tutor produces the REP FOC. Thus, the purpose of REP FOC seems to be to draw the student's attention to the linguistic item under review or examination at the present moment in the tutorial interaction. And, it seems that REP FOCs occur as the tutor entertains a hypothesis that the student has failed to understand or to hear what had been said. That is, the tutor seems to believe that the student has made a mistake rather than an error.

Finally, there are repetitions of entire acts, what we call REP ACT, which involves not only the repetition of the linguistic utterance associated with some manipulation of the simulated world but also requires repetition of the actions taken by the tutor linked to them. REP ACTs seem to be restricted in use to occasions where the tutor believes the student simply has failed to learn something, where he has made what is traditionally described as an error. While ordinarily errors are not directly addressed, but seem to be ignored in favor of new examples for consideration, tutors do occasionally repeat entire acts. We thus can see that REP ACTs are distinct from other forms of repetition in that they address error rather than mistakes or rather than assisting in sustaining a memory representation of the input, but we at present do not know when REP ACTs occur rather than simply using additional and new examples.

These observations and others like them are described in a working paper, a part of which we will present at a major second language acquisition conference in February and hope to publish sometime after that. Most of these insights must still be built into the tutor sub-component of the overall system.

2. The development of model language teaching simulations. During the course of the project we developed a number of listening oriented language teaching simulations. They represent the kinds of simulations we believe will prove effective pedagogically and engaging to the language learner. An examination of these simulations also reveals something of the general nature of simulations for language teaching. Of these simulatio[illegible] [illegible]nly the lifeboat has been implemented fully.

(1) The lifeboat. In this simulation, which was discussed in the Project Description section, the student must provision a lifeboat in response to oral instructions in a second language (either ESL or Japanese). The student moves objects about the computer screen and places them in requested positions in the lifeboat. In order to do this successful, the student must comprehend lexical expressions for the various items and relational expressions for the targeted locations.

The lifeboat simulation has two component lessons. One is a traditional lesson in which the computer tutor controls the interaction, requesting acts of the student, monitoring student performance, and altering in interesting but limited ways the organization of the tutorial according to that performance. The second lesson departs from traditional language teaching activities in interesting and important ways. In this second lesson, which we have called a *prosponsive* lesson, the student manipulates objects in the simulated microworld of the lifeboat scene and the computer tutor responds by describing the resulting state of affairs.

(2) Maptiles. This simulation represents one of our ideal lesson environments. In the development of listening abilities it is quite commonly the practice to use maps of one kind or another to represent a territory through which the student must navigate in response to input in the target language. Maps offer many advantages to the language teacher: (1) maps are good for information transfer and sharing; (2) students find map work engaging; (3) object/place locations and directions given are concrete and observable; and (4) resulting procedural discourses are rich as well as natural.

The maptiles environment provides a toolbox with which teachers can construct their own maps and language lessons based on them. The principal component of the system is the **MAPTILE**, an object composed of three parts: (1) pathways (which define five particular maptile objects), locations (in which building objects may be placed), and (3) buildings (which may have size, address, name, and other internal properties). The maptile inventory is illustrated below.

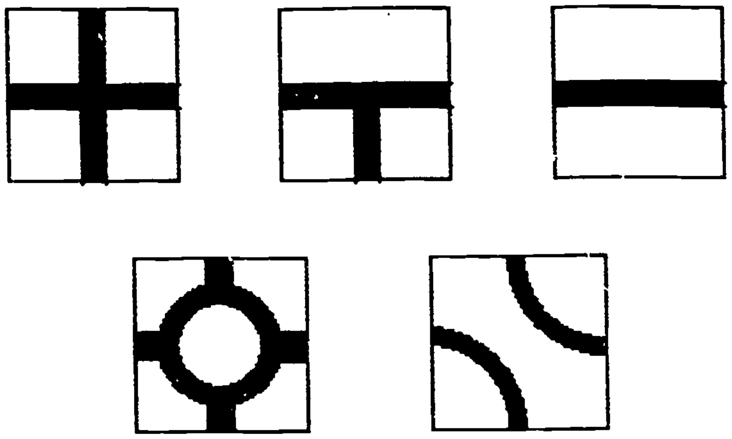


Figure 3. The inventory of maptile objects

In designing a map, the teacher builds a map by selecting maptiles and placing them on the display. Each maptile defines the pathways and other properties exhibited by each maptile selected. Unspecified parameters are assigned default values. For instance, a given pathway might be designated as one-way, if the teacher so desired, or it could be left alone and assigned its default two-way flow of traffic. Having defined a map, the teacher may also place buildings at locations on each tile, defining properties that each building may have (graphic appearance, name, address, hours, and so on).

With a completed map, the teacher can also define a maptile problem. This represents the real world problem the student will be required to manage. Typically, such problems will be defined as **NAVIGATION** problems, in which the student must follow some specified path (as directed by oral input from the computer) under time pressure. The teacher defines the required path, the time limits or other limits on performance, and the language input provided to the student. For example. the teacher might create a simulation in which the student controls a getaway vehicle. The vehicle must be maneuvered to a specific hideout along a particular path by following directions provide as oral input for the student. The student controls the cursor which may be represented graphically as a vehicle or pedestrian or other figure.

(3) **The animated dictionary**. The animated dictionary (AD) is a lexical support system for students and teachers. Students use the animated dictionary to search for lexical items, related items and collocations, and variations from semantic prototypes. Teachers control the extent of student access in a given scenario.

In the animated dictionary, lexical entries for ostensive vocabulary are represented by animated sequences. For example, WALK would show an individual walking across the screen and other manners of locomotion would be similarly displayed (RUN, SKIP, JOG, JUMP, TRIP, etc.). Students have two means of accessing the AD: (1) "meaning" to sound (What is the word for this?), or (2) sound to meaning (What does this mean?).

Dictionary entries include a phonological representation consisting a digitized recordings of the item in a citation form as well as exemplars of its use in context. Th entries also include an orthographic representation, which may be represented along with or independently of the phonological representation or suppressed altogether.

The graphic representation for each entry is composed of prototypical cases along with positive and negative exemplars. Prototypes present stereotypic examples of the requested category. Positive exemplars demonstrate typical and fringe cases. Negative exemplars demonstrate common misconceptions or confusions. In addition to these cases, the AD also links a given entry to parallel and subordinate entries from the same general semantic class.

For instance, RUN is linked to parallel verbs of motion WALK, HOP, SKIP as well as to subordinate semantic associates SPRINT, JOG, LOPE, etc.

The organization of the animated dictionary is a simple hierarchical one in which the student uses buttons on a Macintosh menu to select among alternative actions. The selections take the student to various other menus: BASIC ENTRY (a graphic, animated representation; a phonological representation; an orthographic representation), VARIATIONS (positive and negative exemplars), RELATED ITEMS (semantic associates and contextual uses), and SOUND VARIATIONS (context sensitive variations in pronunciation).

(4) **FlatLand**. In this simulation, the student builds simple two-dimensional configurations from a limited set of objects: black or white, large or small, circles or squares. The tutor initially trains students to deal with the lexical items needed to identify individual objects. It then introduces the specific spatial relations manipulated (*above, below, left, right, between*). Finally, the student is directed to build configurations of these objects like the one shown in Figure (4). This is done by placing one object at a time in its proper location in response to a single, complex utterance in the target language.

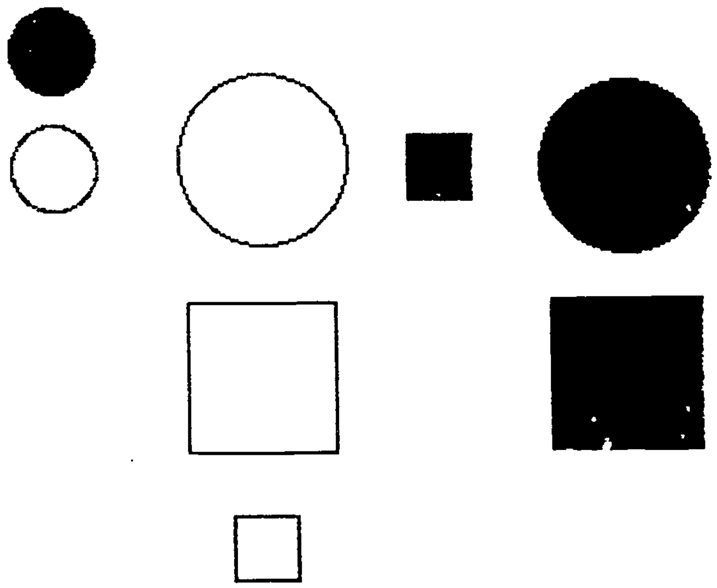


Figure 4. A final FlatLand configuration

(5) **Mystery world**. This simulation places the student in the position of a witness to some set of events. The events are presented as an animated film of some kind or other. Associated with the film is a descriptive sound track which relates the actions witnessed by the student. This narration, whose contents are provided by the teacher, can precede, follow, or occur simultaneously with the movie.

The student engages in two kinds of tasks. First, the student practices ordinary listening, matching the narration with the ongoing events portrayed in the movie. Second, and more interesting, the student can then interact as witness to the computer tutor's detective to solve a mystery presented in the movie. The detective can interview other witnesses in the scene and the student must listen and verify the truth and accuracy of those interview responses. In addition, the detective can interview the student directly, through the judicious use of yes-no questions. This interaction can take the student through the traditional hierarchy of question types in listening: (1) questions of observed fact, (2) questions of inferred fact, (3) questions of inferred motivations, and (4) questions of evaluation.

This particular simulation has been mocked up using Apple animation for one scenario. In this scenario the student/witness observes several people entering and exiting a bookstore. A robbery occurs in the bookstore (unseen directly by the witness), but the timing of events permits the witness to infer who the culprit was.

(6) **Minor simulations.** In addition to the major simulation types described above we also considered and in one case partially implemented several other more minor simulation types. One example is a simulation of the face. By pointing to parts on a human face, the student is able to hear the lexical names for the constituent parts of the face (eyes, nose, mouth ,etc.). This has been implemented.

Other simulations examined included:

(1) ClothesWorld: in which the student travels through a mall to acquire climate appropriate clothing under a limited spending budget.

(2) Plumbing: in which the student builds a water system out of plumbing components in response to instructions from the tutor.

(3) ForestWorld: in which the student manages forest resources according to orally presented information on the state of the forest and possible outcomes of alternative decisions that might be taken at a given moment.

(4) BusinessWorld: in which the student manages a simple manufacturing scenario, buying raw materials, producing manufactured goods, and selling these goods in a competitive market.

It is our belief that these curriculum efforts represent important ways in which the basic computer tutor system might be used effectively. As the project continues, it is our hope that the first four simulations types above will be implemented, both for their own merit and as examples for others to modify and work with.

3. ***The nature of simulations.*** In order to assist teacher in the design of engaging and effective language teaching simulations, we examined the underlying organization of task-oriented language teaching simulations and developed a taxonomy of simulation problem types. These include:

(1) **Resource Management Problems**: the student manages a finite quantity of resources, seeking to maximize their utility under varying constraints and conditions of use.

Examples

Budget management (In order to accomplish some **GOAL**, the student expends portions of a finite **MASS VARIABLE** (money, time, energy)...

Inventory management (In order to accomplish some **GOAL**, the student distributes individual items from a limited **INVENTORY of ITEM TYPES** (auto parts, flora, f una, etc.)...

(2) **Navigation Problems:** the student maneuvers through some medium or territory as directed by the tutor, reaching specified goals and avoiding specified obstacles.

Examples:

MAPTILES: described above.

Adventure simulations: the student travels through a **PHYSICAL WORLD** (dungeon, forest, shopping mall, tourist zone, Paris) collecting **ARTIFACTS** (gold, goblets, butterflies, bric-a-brac, souvenirs, insults) depending on the successful interpretation of **INSTRUCTIONS** (warnings, suggestions, recommendations, etc.) provided by a **TOURGUIDE** (imp, ranger, store proprietor, taxi driver).

(3) **Race/chase Problems:** these represent collocations of limited resource problems (in particular **TIME**) with navigation problems.

4. General principles of communicative language teaching targeting listening comprehension. We effected a search of the pedagogical and theoretical literature in second language learning and teaching and we interviewed practicing second language teachers in order to create an inventory of listening activities. The inventory of activities is organized to reveal: (1) the necessary prerequisite skills required to perform the task, (2) the learning goals for the task, (3) the expected outcomes for the task, (4) the means/methods of presenting/executing the task, and (5) means of evaluating the effectiveness of the task. While no original work was done in this area, it remains useful in organizing suggestions for practicing teachers who might be interested in using the computer tutor system. These insights will be reflected in the final LingWorlds manual.

The LingWorlds Simulation System: Implementation

The LingWorlds tutoring system consists of two component parts. One component is seen and used by the learner to engage in language learning simulations. This component we will call the **tutorial system.** The second component is used by language teachers to create simulations and language learning problems. We will call this component the **authoring system.**

The Tutorial System

The LingWorlds system is illustrated by functional parts in Figure 5. The microworld component is essentially all object-oriented, as is the tutor component. The animator and language generator contain special routines which manipulate digitized data and are written in assembly language to be as efficient as possible. The language generator has a digitized lexicon, and a case-frame semantic grammar, and is capable of dynamically generating utterances from a conceptual representation. It much too primitive to be considered a full-fledged natural language generator but is sufficient for our simple language teaching. A separate generator is needed for each language taught. The animator is a specialized unit for running long animation sequences as "movies" rather than generating them directly in the microworld. However, the microworld objects are capable of simple animation sequences.

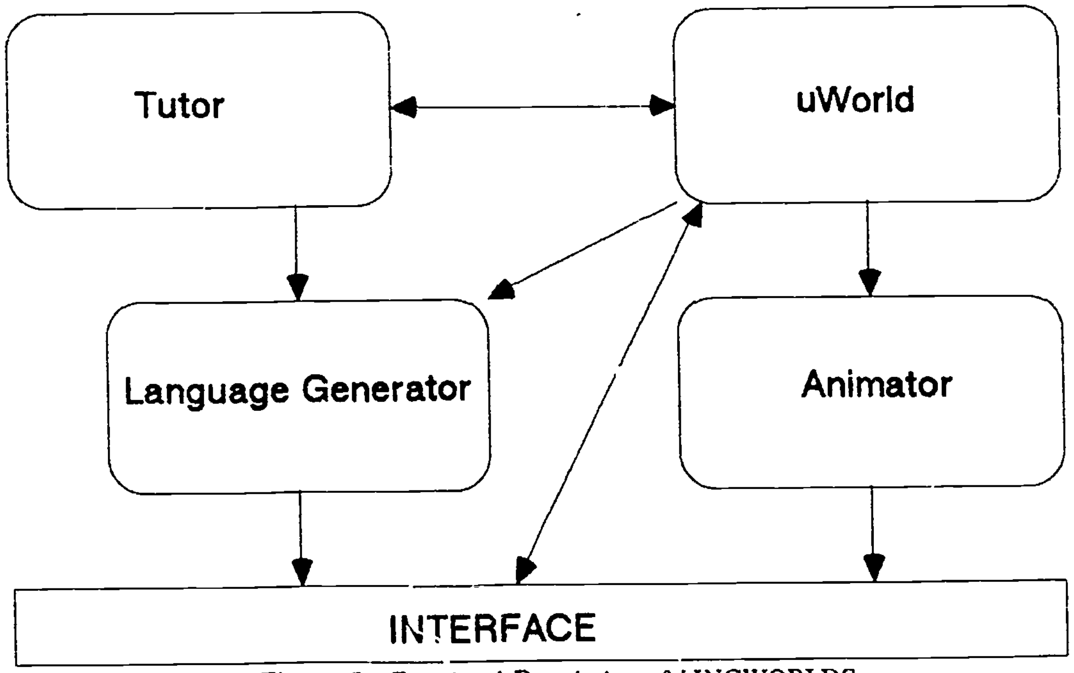


Figure 5. Functional Description of LINGWORLDS

Contexts and Scenes: Each lesson or exercise consists of a sequence of contexts. Each context is either a "movie" sequence or else a "scene". Within a context, a series of movies or scenes may occur. Each scene can be composed of subscenes appearing simultaneously in the same window or in recursive windows. A scene is simply a set of problem-solving objects for the student to manipulate and to focus attention on. A scene may have a background which is simply a bit-mapped picture and not a full-fledged object.

Interactors: Each object represented in a scene is a highly specialized object called an "interactor" object. Interactors in the *Lifeboat* microworld (Figure 1) include the rope, the anchor, the lifeboat, and the question mark (?) icon. Interactor instances have various features, actions and relations with other interactor instances. Interactors are usually visible as a bit-mapped picture and arranged in x-y locations during the initialization of a scene. They can also become invisible. An invisible interactor can mark a spatial region to which other interactors can be moved. Interactors are located on planes with each interactor occupying its own plane. Interactors can swap planes dynamically. This provides the scene with a simulation of three-dimensional reality as interactors are animated by the system or dragged by the user.

Interactor Actions: Interactors can respond to both user actions, such as mouse clicks, and system actions such as message passing. Interactors can highlight themselves and they can speak. System animation for the interactors can result from following any of three methods: an arbitrary path stored by the designer, a computed trajectory, or a location described by a natural language spatial relation, which allows a description of motion by displacement. It should be emphasized that these interactors, while representing concrete entities in the real world, do not manifest the physical laws of that world. Interactors do not "fall" under the influence of gravity, unless caused by the system designer. In other words, the microworld is more of an imaginary, linguistic world. Each interactor responds to 1-click, 2-clicks, press-and-hold, and press-and-drag mouse actions by the user. The designer can choose to enable or disable these methods. Thus the system can enable drag actions on some interactors and ignore them on others.

Semantic Properties: Each interactor has a list of semantic features that define it for linguistic purposes. These features include whether or not the interactor is animate, a person, an artifact, a vehicle, a vehicle of public transportation, or a container. Any other attribute-value pairs can be added by the designer. For example, an author could specify the interactor's color or the lexical item that designates its name. In a Japanese version of *Lifeboat*, animacy features were added to interactors for purposes of generating the correct morphemes during speech generation. Each interactor can also be decomposed into a subset of other interactors. This expresses the has-parts relation. For example a human interactor can be decomposed into further interactors of arms, legs, torso, and head. This relation can also be used for generalized possession as well. Interactors can also have ports which cause auto-message passing between them when they are contiguous.

Spatial Relations: Probably the most difficult and interesting part of the system is the set of spatial relations that we have very thoroughly studied and built into the system. Although this was done initially for the pedagogical goal of teaching spatial relations involving prepositions, it has had interesting side effects for the overall design of the system. For example, it allows interactors to locate themselves in a scene relative to a spatial relationship with another interactors.

Our initial work with spatial prepositions included those in the essentially horizontal and vertical two-dimensional planes: *left of*, *right of*, *above*, *below*, and *between*. (Because of issues hinted at previously, our simulations are primarily two dimensional but because of the possibility of overlapping and movement, they approximate three dimensions. At this time we have yet to implement the depth plane prepositions: *in front of* and *in back of*.)

A major problem for us was where to put the deictic origin of the speaker. Is the speaker, in this case the tutor, describing the scene from the position of the student looking at the scene on the CRT screen, or is the speaker sitting opposite the student? An argument can be made for either case. Certainly the voice is coming from an entity seated across from and facing the student. On the other hand, people using deictic expressions can always see the location of the speaker and make the relative orientation adjustments between what the speaker sees and what the hearer sees. In the case of the computer, one is simply not sure where the speaker is. The least ambiguous assumption is that the descriptions are from the point of view of the speaker in a position of the viewer since the location of the speaker isn't known. Thus spatial relations are computed for a scene from the deictic perspective of the student.

In addition to the issue of point of view, as noted above, it is the case that judgements of these relations can vary depending on shape, size, discourse task, and contextual arrangement. There also may be prototype positions for some situations. We have conducted extensive protocol analyses of human tutors using simple geometric shapes to teach the relations of *left, right, above, below,* and *between* (Tomlin, Douglas, et. al., 1988). Based on these protocols, our own intuitions and psycho-physical experiments, we have developed algorithms to compute these relations in a manner similar to native English speakers (Douglas et al., 1987). These spatial relations algorithms and tutoring strategies have been incorporated into a microworld simulation called *Flatland*.

In order to implement spatial relations in our tutor, we compute on demand for each interactor several spatial properties: center-of-area (centroid), distance, areas that project from edges, and angular displacements. These properties, as implemented in some rather straightforward message-passing algorithms, have sufficed to compute successfully the relations *left, right, above, below,* and a modified two-dimensional *in*. In some linguistic tasks proximity becomes a crucial factor for the computation of these relations and appears as the most difficult component to determine in the general case, since it is clearly dependent on social and psychological factors. We also implement certain spatial properties as features attributed to an interactor. This allows for the specification of a container feature for "in" relations.

Since our primary goal is to allow generalized tutors for various languages, it seems reasonable to allow the teacher/author to change the semantics of spatial relations as they may vary by languages. While this might seem trivial to mono-lingual English readers of this paper, it can be a tricky and difficult problem. For example, imagine a bowl which contains an apple. Both English and Chinese speakers would say "The apple is in the bowl." Now imagine that the bowl is turned upside down (hopefully you imagined the first one in its canonical orientation). English speakers would now say "The apple is under the bowl," (even though it is still within the containment of the bowl), but Chinese speakers would still say "The apple is in the bowl." Clearly the same problems occur in instances like "Get on the bus." in English, which French speakers would imagine as someone on top of the bus. These semantic differences suggest that the knowledge representations for spatial relations will have to be open to change by the authoring system. This could be an exceedingly difficult goal to achieve.

Integrating the Microworld with the Tutor

The preceding description or the microworld components describes how program code is modularized by what is essentially display and control of the interface. Since much of this is generalizable, it is highly productive to have them available as part of the class definitions. However, as just described, the basic control appears to be primarily student initiated. If the student clicks with the mouse on a particular interactor on the screen, a method is appropriately activated which may move the interactor, cause it to say something, etc. This is the basic flavor of all exploratory learning environments. It is the case, though, that we often want to introduce more teaching intervention into student actions. Thus, ICAI systems vary along a continuum from totally event-driven exploratory environments, to coaches which are embedded within a game structure, to goal-directed tutors which have a highly specified control structure. Teaching expertise includes how to tutor, what instructional approach to use, and why and how often to tutor the student. Insights into the complexity of language tutoring are described below as well as in Douglas (in press) and Tomlin, Douglas, et. al. (1989).

Control from the tutor is introduced into LingWorlds by the tutor object. In LingWorlds the control is knit together by message-passing between interactor objects and the tutor object. A totally exploratory microworld has its control locally defined with the behavior of each object tied to student actions. A game microworld increases tutor control. Finally, a goal-directed tutor microworld controls most of the interaction through a task-based agenda. Even in the goal-directed tutor, interactors still retain individual control over the semantics of their own actions. For example, the tutor can be notified that a particular interactor has been dragged, and through additional message-passing with the interactor determine the location. Thus, the object-oriented paradigm allows an event-driven format that accommodates user-initiated actions as well as internally controlled actions.

The Authoring System

Characteristics of Authoring Systems

Authoring lessons for computer-based second-language instruction is, traditionally, a time-consuming and intensive task which has all of the problems of interface construction. With the advent of bit-mapped displays and mice, the author is faced with ever increasing design complexity as graphics and sound supplement text displays, windows allow multiple contexts for user tasks, and pointing devices join keyboards. On sophisticated systems such as Interlisp on the Xerox 1100's, the interface programming effort has been informally estimated to consume about 80% of the total programming time (Smith, 1984). The long delay experienced in software development for even less complex machines such as the Macintosh can similarly be attributed to the complexity of composing over 600 ROM-based interface functions into usable interactive programs.

The reasons for this increased complexity are due to several factors. First of all, designing an interactive interface is not an exact engineering task, and much less a science. Although the demands for understanding human-computer interaction have been met with an explosion of experimental psychology literature, it still remains that knowledge of human behavior and language is not a precise science. Consequently, the author must still rely upon a generate and test methodology that introduces users into the design loop early in the process. A second reason that programming complexity has increased is that much of the output of the program is now represented visually. This requires programming of graphic algorithms, perhaps even animation. These techniques are not well-known by the average programmer and need constant visual verification during the programming process. Thirdly, the coding effort for implementing the details of pull-down menus and user-definable windows is enormous. While many systems have window managers to assist in the run-time association of mouse and keyboard events to particular windows and menus, the author may still have to handle much of the control in the application code. Finally, the aspects of multiple window contexts which are dynamically generated by the user at run-time contribute to an event-driven program that is not easily represented in standard procedural languages such as PASCAL. In classical procedural languages, input and output is primarily from files. The state of the program determines almost entirely the state of the output. Interactive interfaces require that the state of the interface as well as the state of the program be taken into account. This leads to event-driven rather than data-driven programming (Shaw, 1986). Even standard approaches to formal specification of interfaces with state transition diagrams (cf. Jacob, 1985) lose their usefulness as the sequential nature of text-style dialogue interaction disappears to be replaced by direct manipulation.

Although there has been a good deal of research on reducing the complexity of interface programming by developing User Interface Management Systems (UIMS), these research efforts fail to sufficiently address the programming design effort by concentrating instead on the relationship between the low-level functionality of the workstation and the specification of various interface attributes in a particular language. (See, for example, Buxton et al., 1983; Flecchia & Bergeron, 1987; Hayes et al., 1985; Hill, 1987.)

What few systems have been developed for interface design, such as Trillium (Henderson, 1986), AIDE (Hix & Hartson, 1986) and PANTHER (Helfman, 1987), tend to have limitations caused by specializing in a particular type of interactive format. For example, Trillium is used to design copiers and AIDE, dialogue interaction. The limitations make it difficult to generalize these systems to the creation of microworlds for linguistic interaction.

Design is a highly knowledge-intensive activity which results in the creative production of an object or process. Most theories of design tend to emphasize top-down refinement beginning with a specification and proceeding to the implementation. This prescriptive approach ignores many of the bottom-up practices of working designers. For example, human designers are more opportunistic and domain knowledge intensive, such as using an existing design to guide the design process. Adelson and Soloway (1984) in their study of software designers have also highlighted the usefulness of simulating the specification as an attempt to ferret out possible specification bugs. Finally, implementation imposes many constraints on specification. Thus, the prevailing utilization in practice (as opposed to theory) of what are called rapid prototyping methods for design. Therefore, it is important to let the author of a language lesson quickly build and then refine the microworld in which the tutoring interaction will take place. Tools for the creation of graphics and sound already exist; what is lacking is an environment which easily provides programmatic functionality for these elements.

An important result from studies by Adelson & Soloway (1984), Kant (1985), Kant & Newell (1982), and Steier & Kant (1985) in the domain of software engineering is the following:

(1) Designers rapidly develop a kernel idea and refine it during the design process.

(2) Designers spend about half of their time simulating the behavior of their programs. The simulation process serves many functions: it helps the designer integrate constituents from several parts of the design; it serves as a kind of agenda to keep track of subtasks that require attention; it encourages a kind of balanced, methodological refinement of the software system; and it allows for comparison to the design goal. Simulation helps the designer identify interesting opportunities for improving the design.

(3) Designers take both mental and written notes on things to remember later in the design such as constraints, partial solutions, and potential inconsistencies. These were not handled immediately since they were at a greater level of detail than the current state of the design. In practice, the designer is frequently able to avoid problem solving the entire design by recalling previous partial solutions

(4). This suggests that a support system for design should provide examples of common parts and previous designs which can be cannibalized. This supports the idea of inheritance through class specialization or prototype-modify found in object-oriented languages.

It has also been observed that designers make mistakes which they do not recognize until far along in the actual implementation process when change may have a radical effect on previous design choices. This suggests that simulation of the design as early as possible is crucial for debugging it . In a nutshell, this is the justification for techniques of rapid prototyping. For mechanical engineers drawings play an important role in the process of design. They act as completeness checks, simulation, and analysis. Interface designers no doubt gain the same value by direct graphic layout of the interface objects. This supports the notion of a direct manipulation approach to design. The objects which are to be the context and referents of the linguistic interaction should, then, be manipulable by the lesson author in much the same way that objects can be manipulated in the microworld itself.

We believe that much of the effort and complexity involved in lesson design and implementation are be reduced by the introduction of rapid prototyping using object-oriented programming. In addition, we believe that a system for design of the lesson interface should provide a specialized environment for this aspect of programming. Furthermore, we believe that this UIDS should incorporate, wherever possible, direct manipulation techniques as the fundamental presentation to the author. The system that we have developed thus incorporates two programming methodologies, direct manipulation and object-oriented programming.

Object-oriented Programming
The message-passing nature of object-oriented programming languages, exemplified by languages such as Smalltalk, Lisp with Flavors, and NEON, an object-oriented Forth, allow easier representation of the event-driven nature of the modern interface. Class inheritance increases programming productivity by allowing design by taxonomic classification and specialization, with less duplication of common code. Multiple copies of a defined object can be easily and dynamically instantiated to further reduce programming code. Thus, development time can be reduced by a factor of four or five (Schmucker, 1986). Lesson authoring becomes an effort of selecting from a library of reusable programming parts. This encourages consistency across lesson design. The encapsulation of functions within objects as cleanly interfaceable modules reduces programming bugs.

Although all object-oriented systems offer an extensive collection of interface "pieces" such as a window package that the programmer can use to build the interface, only Smalltalk has a well-developed and systematic approach to the user interface incorporated into a system called the Model-View-Controller (MVC). This system provides the programmer with ready-made text editor, scrolled windows, pop-up menus and process scheduler. Unfortunately, the MVC is not documented in any of the three currently available Smalltalk books, a fact which makes it unavailable even to Smalltalk programmers.While the Model-View-Controller holds promise as a viable system for interface design, our laboratory experience with it has shown it to be less than desirable. The separation between these three elements is not often easily made, i.e. controllers have views, and view information is frequently still embedded in the model (the application code). A spatial mapping between the displayed object (view) at which the user may point and the model object does not exist. The lack of an extensive collection of interface functions such as dragging and simple animation. Finally, all since views are dependent upon receiving change messages from a model, an inefficient synchronous system of sending messages to all views is required. While several object-oriented systems such as STEAMER (Hollan, 1986), ThingLab, and ARK (Smith, 1986) provide some unusual approaches to modifying the user interface, none are as extensive as a Smalltalk system called Programming by Rehearsal (Gould & Finzer, 1984). It was developed for non-programming teachers to build simple educational games. Programming by Rehearsal does not use the Model-View-Controller system of Smalltalk, but substitutes its own interface model whose metaphor is that of a theatrical production. Using a primarily visual programming environment "performers" (objects) can be moved around on "stages" (windows) and taught how to interact with each other by sending "cues" (messages). Performers have simple animation and picture display as part of their definition.

Class inheritance is finessed by using the concept of prototype. Thus, the basic definition is to copy an existing object and then modify it, rather than define a sub-class and declare an instance.

Direct Manipulation

Object-oriented languages usually encourage a rapid-prototyping style of program development that allows incremental visual verification of program modules by the programmer. The programming environment is richly endowed with integrated editors, browsers and debuggers. However, interface design is done through traditional textual programming, not by a form of direct manipulation. Direct manipulation environments provide the user with several highly useful features when performing a complex task: continuous representation of the object of interest, physical actions or labeled button presses instead of complex syntax, and rapid incremental reversible operations whose impact on the object of interest is immediately visible (Shneiderman, 1982).

The incorporation of direct manipulation interfaces into authoring systems (and programming generally) has taken two forms: visual programming and programming by demonstration. The former emphasizes the visual representation of the code (Myers, 1987). In describing an interface specification system, Jacob (1985) recommends the use of graphically displayed state transition networks as design notation over BNF grammars because of user interface ease, emphasizing the importance of visible graphics to represent abstract entities over linguistic description. Programming by demonstration, emphasizes the physical actions of the programmer to capture the actions of the user at the interface (Myers, 1987; Gould & Finzer, 1984).

A direct manipulation design environment for programming the interface allows the designer to specify the interface both visually and by manipulation in order to automatically generate the code. For example, a window can be built out of window parts with sizing done by manipulation rather than textual specification. For design in areas where specification languages may be difficult to implement (for example, graphic design) or where it is difficult to formally specify the efficient implementations (for example, human computer interaction), direct manipulation rapid prototyping systems appear preferable to textual parameters. A specification of the user's possible physical interaction can be done by simulating that interaction (Myers, 1987).

How The LingWorlds Authoring System Works

Following the precepts delineated in the preceding section, we developed the LingWorlds Authoring System. Briefly, the goals of the authoring system were to enable a non-programming teacher of language to build microworld-based lessons. The system integrates images, sounds, and interface functionality in an authoring system for second-language lessons. The authoring system's principal characteristics include object-oriented prototyping, making interface-related functionality available to all objects, and providing a language for object functions powerful enough to encompass domain functions in the interface. Using the authoring system, functioning lessons can be developed, tested, and refined.

System Overview

The LingWorlds authoring system is an interpreter implemented on the Macintosh II computer in Allegro Common Lisp using Allegro's Object Lisp system. It generates Lisp code which can be supplemented by additional programming. The system is built from a set of primitive features, which are either constructed in the authoring system (such as text) or are imported from other programs (such as graphic images). These primitives include digitized sound, Macintosh-style images, locations, integers, booleans, text, and icons.

"Tutor" menu permits the user to create and develop the tutor in the Aether window. Finally, the "Windows" menu lets the user toggle between the World and Aether windows. Many of the menu choices are shadowed by mnemonic Macintosh-style command-key combinations.

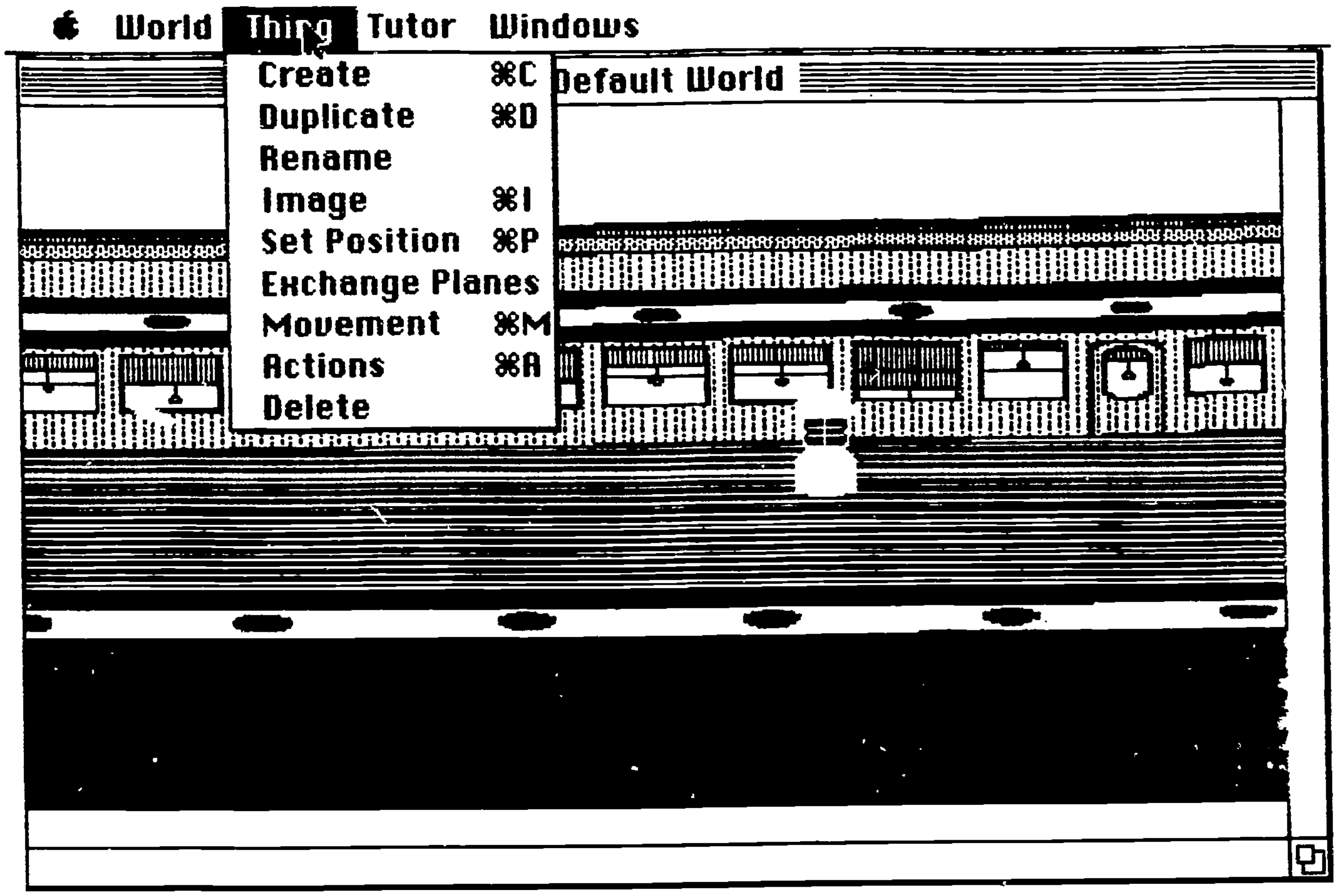
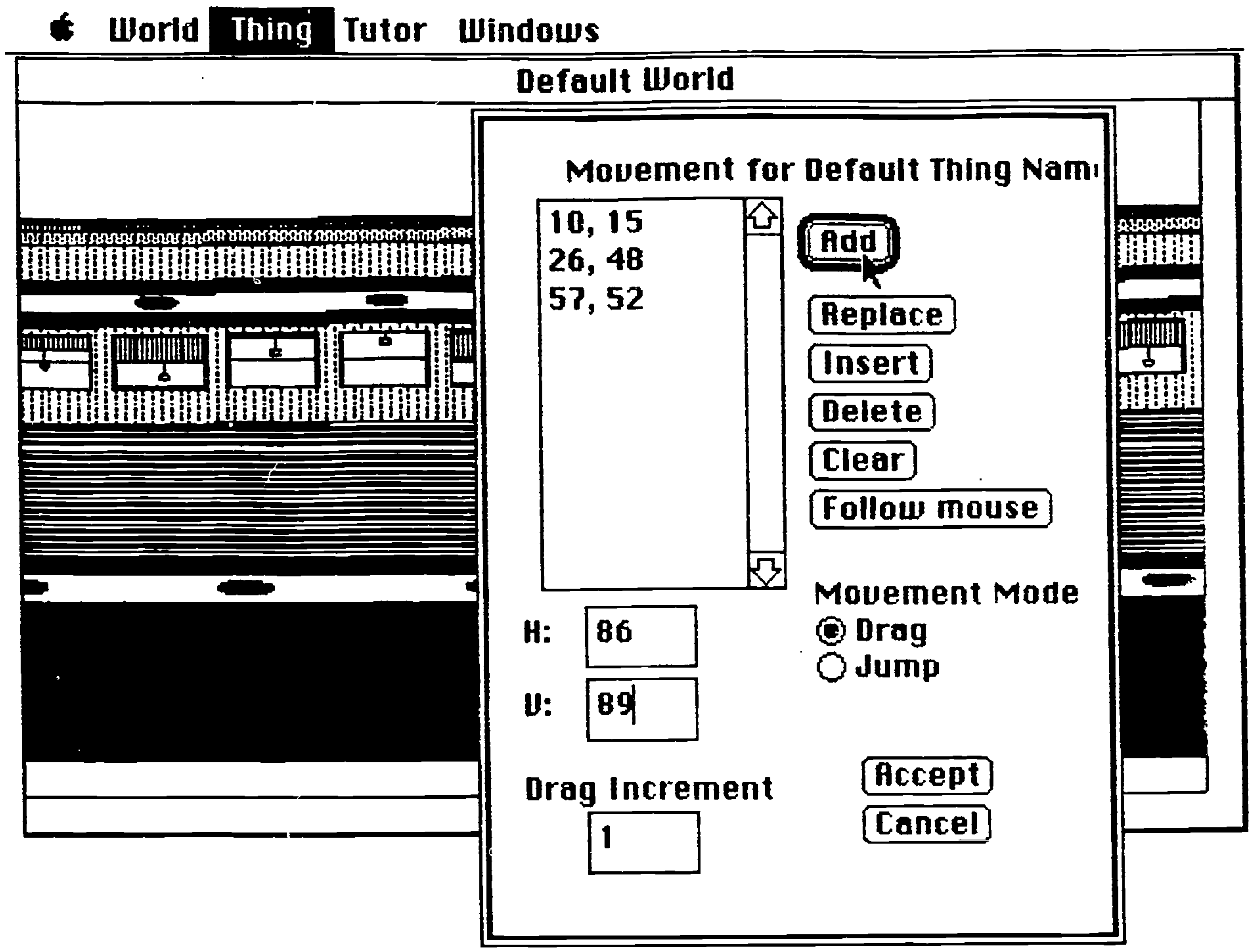


Figure 7. Menu Selections for Things

LingWorld's things are prototypical rather than classed. That is, every thing is created with all the characteristics of the basic prototypical thing. There is no notion of inheritance or class-based specialization. The various aspects of things are determined by a user through direct manipulation, menu-based construction, or in an English-like language using structured dialogues.

Direct Manipulation

Direct manipulation is used to determine positioning and paths of things. The user can simply drag a thing to a desired location in the window. Things can be displayed in different forms, including image, icon and text. Things not only have the static notion of location but a related dynamic notion of path as well. Each thing can automatically record and then follow a sequence of positions. Such paths can either be recorded demonstratively by dragging the thing with the mouse, or by specifying exact locations. In both cases, the path can be edited using a suitable dialogue. Figure 8 shows the author editing a thing's path. Multiple things can follow their paths simultaneously to produce simple animations.



10, 15
26, 48
57, 52

H: 86

V: 89

Drag Increment 1

Movement Mode: [✓ Drag] Jump

Figure 8. Editing a Thing's Movement Path

Menu-Based Construction

Menu-based construction is used for specifying the thing's graphic representation. Each thing can have a graphic image, a textual representation, and an iconic representation, although only one of these can be displayed at a time, of course. Thing menu commands allow the user to choose an image and a mask for the graphic image representation, to set the text for the textual representation, to select an icon for the iconic representation, or to choose the displayed representation of the thing from any of these. Menu-based construction is also used in building the lexicon, creating and deleting things, and setting attributes of things.

Each thing is created with a set of seven built-in attributes. Attributes are analogous to instance variables in regular Lisp things, but are strongly typed. Each attribute has an initial and a current value, and is defaulted appropriately. The built-in attributes are *position*, which is the location of the thing in the Interface window; *highlighted?*, a boolean value indicating if the thing's representation is in reverse video; *visible?*, a boolean value indicating if the thing's representation is visible or invisible (invisible things are useful for things like "hot spots" in larger things); *draggable?*, a boolean value indicating if the thing can be dragged when the interface is in running mode; *click 1?* and *click 2?*, boolean values indicating if the thing will execute its response to being clicked or double-clicked in running mode; and *plane*, an non-negative integer indicating the foreground/background position of the thing's representation relative to other things in the window. New attributes may also be created by the user, who must specify the appropriate type.

Adding Functionality to Things

Related Languages

In developing the action language for LingWorlds, we looked at a number of user interface design languages, including ExperInterface Builder (for ExperCommonLisp), a system developed by Luca Cardelli (Cardelli, 1987), Hypercard, and Programming by Rehearsal (Gould and Finzer, 1984).

ExperInterface Builder is built on top of Macintosh ExperCommonLisp and was developed by Jean-Marie Hullot. It allows the designer to select interface objects (buttons, text box, scroll bar, etc.) from a palette similar to MacPaint and drag them to the desired location in a window. Each object is associated with a Lisp function that will be evaluated when a user event occurs. Graphic images can be imported from MacPaint to give individualized appearance to the interface objects. Testing and debugging can be rapidly done since the interface design is entirely within the ExperCommonLisp environment.

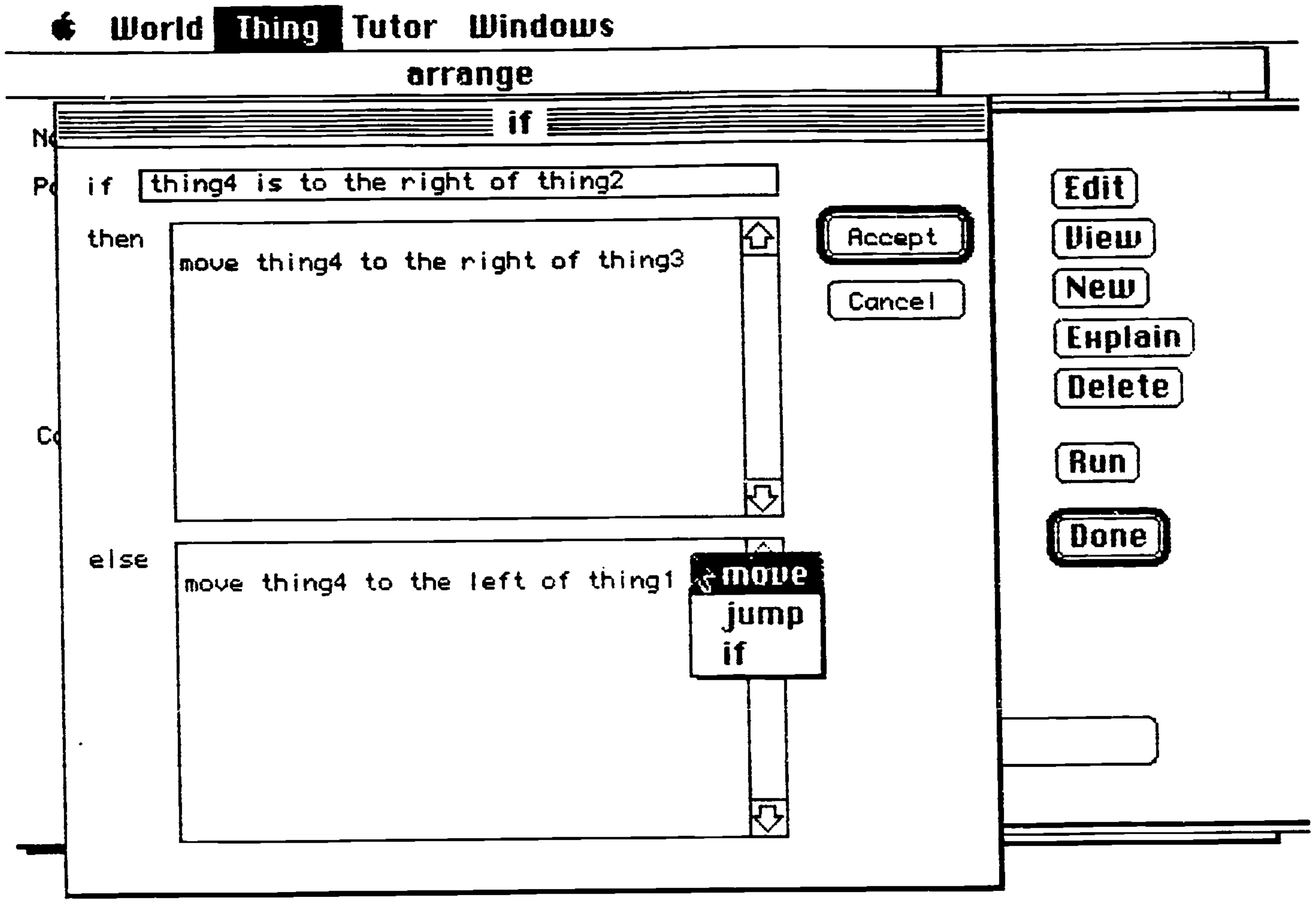
The system developed by Luca Cardelli is similar to ExperInterface Builder but allows a more general concept of user interface objects called *interactors* which are located within a higher level object called a *dialog*. Each interactor's appearance is customizeable by location, size and graphics. Interactors can be composed into groups. Interactors communicate with the application by way of an abstraction of user information called the *events*. Events are defined as having attributes, state, and status. This allows the application to avoid low-level attention to mouse and keyboard actions. The interface designer has available a dialog editor which allows mouse-based selection of interactor instances and customizable appearance. Cardelli's system was built for use with Modula-2+ and runs on the Firefly personal workstation.

Hypercard is a system recently developed and released by Apple Computer for the Macintosh. It is intended as a programming system for novices. The system is object-oriented and based on ObjectPascal. What is exciting about it is that it provides a direct manipulation interface design system which is very easy to use. There are six major kinds of objects: stack, card, button, field, background and picture. The first five are first-class objects, in that they can have scripts containing message-handlers. Picture is a second-class object which can display an image but does not allow user interaction. (Note that this differs from ExperInterface Builder.) Cards are objects that are linked to form a tangled hierarchy. They are essentially a window and can contain the other types of objects. The script writing language, Hypertalk, is a combination of Pascal and Smalltalk and has about 40 basic commands and a reasonable set of control structures.

Action Language in LingWorlds

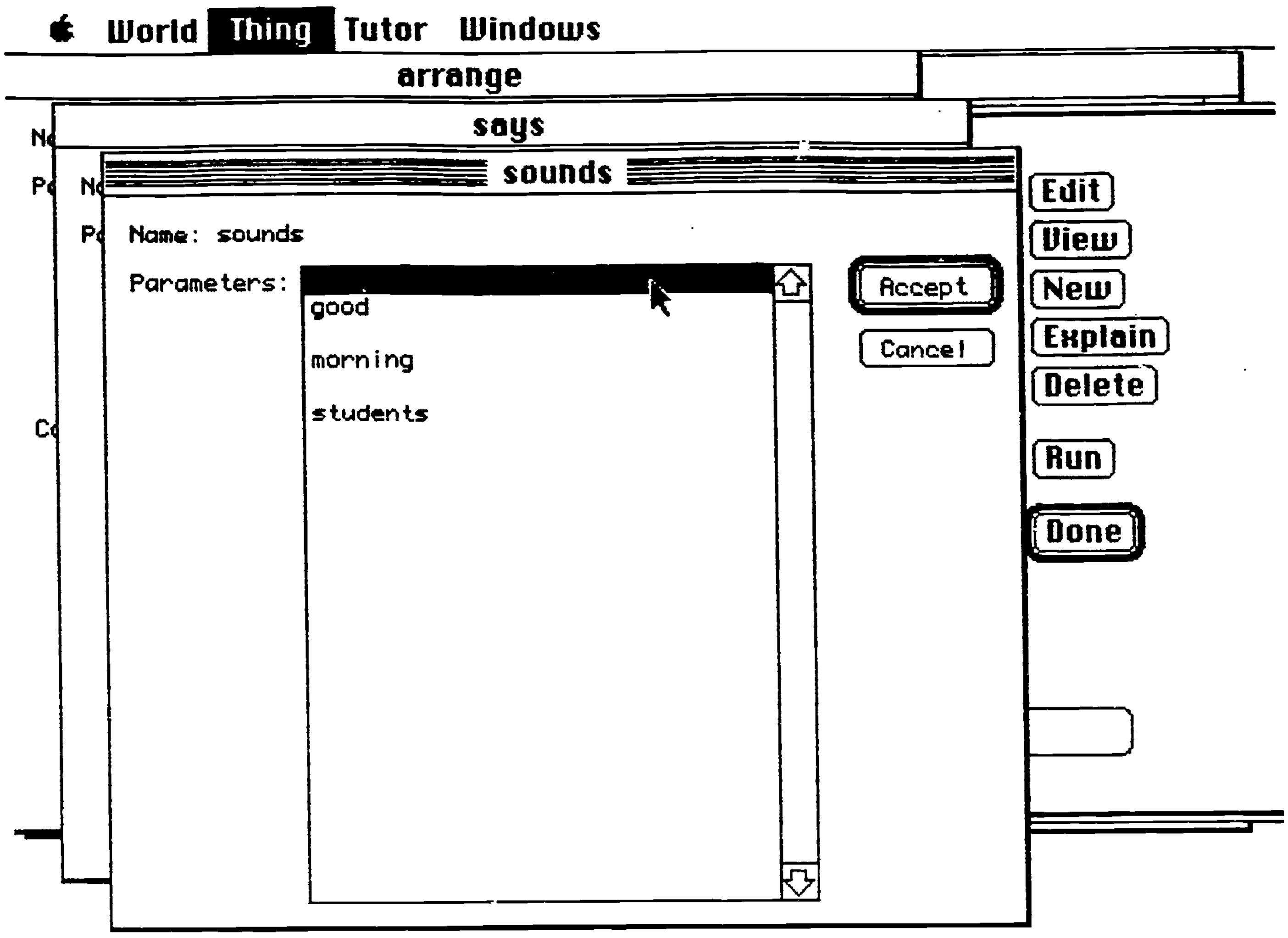
The LingWorlds authoring system's English-like language for methods is used to create running-mode functionality for things. Things have a (growing) set of primitive actions, such as *flash*, *highlight*, *say*, and *move*. There are also control primitives such as *if* and *repeat while*. From these primitive actions, users can also compose additional actions. Taken together, the built-in primitive and user-composed actions are available to all things and are called "global actions." Each thing also has a set of six "local actions," which are functional responses of the specific thing to run-time events. The local actions are *1 click action*, *2 click action*, *dragging action*, *landing action*, *evaluation action*, and *history action*.

The actions, both global and local, are written by the user in a semantically based structure editor using hierarchical dialogues. There is no typing of text, simply selection of pop-up menu items. The system makes sure that the menu items presented to the user are syntactically and semantically appropriate. Thus the LingWorlds authoring system differs from Programming by Rehearsal in that LingWorlds 1) does not provide for writing methods by "watching" the user, but 2) does provide a high-level, semantically appropriate language for construction of actions rather than require the user to program in the underlying language in which the system was implemented. In Figure 9, the author is expanding the template for an "If-then-else" statement.



```
if    thing4 is to the right of thing2
then  move thing4 to the right of thing3
else  move thing4 to the left of thing1
```

Figure 9. Programming an "If-then-else" Statement



Name: sounds

Parameters:

good

morning

students

Figure 10. Programming a "Says" Statement

Figure 10 shows the author constructing an action in which the thing says the phrase "Good morning students."

The tutor is also a thing. Actions can be called and written, using the same interface, for the tutor to produce appropriate direction of the lesson. Direct selection of tutoring modes is also enabled.

Using the Authoring System

Without actions, things merely have static qualities. These qualities are represented by, for example, the thing's graphic representation, its name, and its attributes. Having presented the various parts of the LingWorlds authoring system, we now discuss the construction of things with four distinct kinds of functionality. We note that the functionality of things in LingWorlds reflects a set of levels of behavior inherent to interfaces: behavior 1) affecting only the thing itself, 2) affecting other things, 3) taking state into account, and 4) as a group or aggregation.

Actions Limited to Self: In a simple exploratory version of the lifeboat lesson shown being built in Figure 1, the lantern was set up so that it spoke its name when it was clicked. The attributes of the thing were set so that 1-click? was True and draggable? was True. The thing's 1 click action was constructed to be *flash self; say "The* *lantern."* Thus when, in running mode, when the user clicks the mouse once on the lantern thing, it will flash itself and cause the phrase "The lantern" to be uttered. The attributes and behavior of no other thing are affected.

Actions Affecting Other Things: Things can also have actions which affect each other. Let us suppose that the intent of the author is to have some things cause the lifeboat to flash whenever they are dragged. To accomplish this, the author sets the thing's landing-action? attribute to True and then constructs the thing's landing-action local action as *flash lifeboat*. As a result, any time the student stops dragging a thing with these characteristics, the lifeboat's image will flash.

State-Based Actions: A more complex set of actions takes into account the history of the user's interaction. Thus, for example, the author might wish to provide a history-based tutoring strategy. One way of doing this would be to create a new thing-attribute like "Number of wrong tries." The tutor, then, in directing the student to perform a task in the microworld could tailor its responses based on the value of this attribute for a particular thing. Thus if the value of the wrong-tries attribute for the lantern were zero, the tutor might simply repeat the instruction. If the value were sufficiently large, the tutor might demonstrate the requested task for the student.

Group-Related Actions: Finally, we look at the behavior of explicitly grouped things. For example, the lifeboat thing consists of two things: the lifeboat and its "inside," which corresponds to the lifeboat's floor. The inside is used for determining if another thing is in fact "in" the lifeboat. Obviously, much more complex aggregations and behaviors can be obtained by suitable modification of the action of the constituent objects which compose the group. The important thing to note here is that the objects actions directly encode their interface functions. In a sense, LingWorlds objects represent a "deconstruction" of the usual interface features. That is, the functional limitations of stereotypical interface components have been removed; in their place are the members of the basic function set of LingWorlds prototypical objects. From these elements, new microworlds are constructed.

Implementation

LingWorlds is currently implemented fully on a Macintosh II. The Macintosh version is written in CommonLisp. The movies and digitized sound are files created by existing commercial products to which we had to program some conversion software to load them into the Macintosh resources. All the actual code for the interactors and tutor objects are also loaded as resources. This gives us a "declarative" feel to the object-oriented code and reduces the complexity of loading in the program as part of the object-oriented environment.

Evaluation

Our informal evaluation of the system indicates that the average time to prototype a microworld is about several orders of magnitude less the time taken by Pascal programmers on the Mac (days rather than months). We have yet to extensively test the system with more formal evaluation studies, but those are planned for the future. Since the system is actually quite a powerful system for creating *any* instructional software, a version of the system was recently requested and sent to Yale University for use in building instructional software for mathematics tutoring.

We have built both an English and Japanese version of *Provisioning the Lifeboat*, with all the expected savings in programming. We built the exploratory version first, and then added the directed tutor version. A game version has not been programmed, but would be trivial. Since interactors and tutors can be specialized for any particular microworld. most of the code is inherited and reusable. The Japanese version of the *Lifeboat* is virtually identical to the English except for the addition of animacy features required for speech synthesis. Instantiation allows easy copies of objects with the minimum of programming effort. We have also built a microworld called *Flatland*, in which the student is taught geometric shapes, colors, size and spatial relations. This is a directed tutor version, based on extensive protocols with human tutors. The implementation of *Flatland* took one day, since almost all of the code was reusable from the *Lifeboat* problem. We have designed several other microworlds of various types, as described above.

F. Summary and Conclusions

In this report we have attempted to describe the principal outcomes and problems associated with our project to build an interesting and effective computer-based second language tutor, LingWorlds. Over the course of the project, we found that the basic problems of interest to us in creating this system increased in complexity and scope. However, the end result of the project is the creation of a ICAI system that teachers will be able to use coupled with important new insights into the nature of language learning and teaching.

References

Adelson, B. and E. Soloway. 1984. A cognitive model of software design. Report No. 342, Department of Computer Science, Yale University.

Anderson, J.R., C.F. Boyle, and B.J. Reiser, "Intelligent Tutoring Systems," *Science*, Vol. 228, April 26, 1985.

Asher, J. 1966. The learning strategy of the total physical response: a review. *Modern Language Journal* 50: 79-84.

Asher, J. 1969. The total physical response approach to second language learning. *Modern Language Journal* 53: 3-17.

Asher, J. 1977. *Learning Another Language Through Actions: The Complete Teacher's Guidebook* Los Gatos, CA: Sky Oaks Publications.

Brandt, M. 1984. *Intending and Acting: Toward a Naturalized Action Theory.* New York: MIT Press.

Burton, R.R. and J.S. Brown. 1982. An investigation of computer coaching for informal learning activities. In D. Sleeman and J.S. Brown (eds.), *Intelligent Tutoring Systems.* London: Academic Press.

Buxton, W., Lamb, M.R., Sherman, D., and Smith, K.C. 1983. Towards a comprehensive user interface management system. *Computer Graphics* 17.3: 35-42.

Cardelli, L. 1987. Building user interfaces by direct manipulation. DEC Systems Research Center Technical Report #22, October 2, 1987.

Clancey, W.J. "Tutoring Rules for Guiding a Case Method Dialogue," in D. Sleeman and J.S. Brown (eds.) *Intelligent Tutoring Systems*, Academic Press, London, 1982.

Douglas, S.A. 1986a. Prospects for an ICAI authoring system: a review of computer-based design. *Proceedings of the IEEE Systems, Man, and Cybernetics Conference*, Atlanta, GA, 1986.

Douglas, S. A. In press. Detecting and repairing tutoring failures. In P. Goodyear (Ed.) Tutoring Knowledge and Intelligent, Ablex, in press. Also available as Dept. of Computer and Information Science, University of Oregon, Technical Report CIS-TR 88-09, 1988.

Douglas, S. A., Novick, D. G., and Tomlin, R. "Consistency and variation in spatial reference." *Proceedings of the Ninth Annual Cognitive Science Conference*, July 1987.

Flecchia, M.A. and Bergeron, R.D. 1987. Specifying complex dialogs in ALGAE. *Proceedings of the Human Factors in Computing Systems and Graphic Interface: CHI + GI 1987*, Toronto, Canada, April 1987.

Gould, L. and W. Finzer. 1984. *Programming by Rehearsal.* Technical Report. #SCL-84-1, Xerox PARC, Palo Alto, Calif.

Hayes, P.J., and Szekely, P. 1985. Design alternatives for user interface management systems based on experience with COUSIN. *Proceedings of the Human Factors in Computing Systems: CHI' 85*, San Francisco, CA, April 1985.

Helfman, J. 1987. PANTHER: A specification system for graphical controls. *Proceedings of the Human Factors in Computing Systems and Graphic Interface: CHI + GI 1987*, Toronto, Canada, April 1987.

Henderson, D.A. 1986. The Trillium user interface design environment. *Proceedings of the Human Factors in Computing: CHI'86*, Boston, MA, April 1986, 221-227.

Hill, R.D. 1987. Event-response systems - A technique for specifying multi-threaded dialogues. *Proceedings of the Human Factors in Computing Systems and Graphic Interface*, Toronto, Canada, April 1987.

Hix, D. and Hartson, H.R. 1986. An interactive environment for dialogue development: its design, use and evaluation. *Proceedings of the Human Factors in Computing: CHI'86*, Boston, MA, April 1986.

Hollan, J., E.L. Hutchins, T.P. McCandless, M. Rosenstein, and L. Weitzman. 1986. *Graphical Interfaces for Simulation*, ICS Report 8603, Institute for Cognitive Science, UC San Diego, La Jolla, Calif., May 1986.

Jacob, R.J.K. 1985. A state transition diagram language for visual programming. *IEEE Computer*, Vol. 18, No. 8, August 1985.

Johnson, K. 1982. *Communicative Syllabus Design and Methodology*. Oxford: Pergamon Press.

Kant, E. 1985. Understanding and automating algorithm desing. In *Proceedings of IJCAI-85*, Los Altos, CA: Morgan-Kaufmann. 1243-1253.

Kant, E. and Newell, A. 1982. Naive algorithm design techniques: A case study. In *Proceedings of the European Conference on Arttificial Intelligence*, Orsay, France, July. 40-51.

Krashen, S. 1977. *Second Language Acquisition and Second Language Learning*. Oxford: Pergamon Press.

Krashen, S. 1982. *Principles and Practice is Second Language Acquisition*. Oxford: Pergamon Press.

Krashen, S. and Terrell, T. 1983. *The Natural Approach*. San Francisco: Alemany Press.

Myers, B. 1987. Creating dynamic interaction techniques by demonstration. *Proceedings of the Human Factors in Computing Systems and Graphic Interface: CHI + GI 1987*, Toronto, Canada, April 1987.

Papert, S. *Mindstorms: Children, computers and powerful ideas*. Basic Books, New York, NY, 1980.

Postovsky, V. 1977. Why not start speaking later? In M.Burt et al (eds): *Viewpoints on English as a Second Language*. New York: Regents.

Postovsky, V. 1979. Effects of delay in oral practice at the beginning of second language learning. *Modern Language Journal* 58:229-239.

Schmucker, K. 1986. MacApp: An application framework. *BYTE* 11.8.

Schumann, J. 1978. Social and psychological factors in second language acquisition. In J. Richards (ed.), *Understanding Second and Foreign Language Learning: Issues and Approaches*, 163-178. Rowley, MA: Newbury House.

Selinker, L. 1972. Interlanguage. *International Review of Applied Linguistics* 10: 209-231.

Shaw, M. 1986. An input-output model of interactive systems. *Proceedings of the Human Factors in Computing Systems Conference: CHI '86*\ Boston, MA, April 1986.

Shneiderman, B. 1982. Direct manipulation: a step beyond programming languages. *IEEE Computer* 16.8: 57-69.

Sleeman, D., "Assessing Aspects of Competence in Basic Algebra," in D. Sleeman and J.S. Brown (eds.) *Intelligent Tutoring Systems*, Academic Press, London, 1982.

Smith, R.G. 1984. On the development of commercial expert systems. *AI Magazine* 5.3: 61-73.

Smith, R. 1986. The Alternate Reality Kit. *1986 IEEE Computer Society Workshop on Visual Languages*, July 1986.

Soloway, E., B. Woolf, P. Barth, and E. Rubin. 1981. "MENO-II: An Intelligent Tutoring System for Novice Programmers," *Seventh Int'l Joint Conf. Artificial Intelligence*, Vancouver, Canada.

Steier, D. M. and Kant, E. 1985. Symbolic execution in algorithm design. In *Proceedings of IJCAI-85*, Los Altos: Morgan-Kaufmann, 225-231.

Tomlin, R.S.; Douglas, S.A.; & Novick, D. 1988. Modeling the meaning and use of spatial relations. Unpublished ms. Department of Linguistics, University of Oregon.

Tomlin, R.S.; Douglas, S.A.; & Novick, D. 1989. The microanalysis of individual tutorials. Unpublished ms. Department of Linguistics, University of Oregon

Widdowson, H.G. 1978. *Teaching Language as Communication*. Oxford: Oxford University Press.

Widdowson, H.G. 1979. *Explorations in Applied Linguistics*. Oxford: Oxford University Press.

Winitz, H. (ed). 1981. *The Comprehension Approach to Foreign Language Instruction*. Rowley, Massachusetts: Newbury House.

Winitz,H. and Reeds, J. 1973. Rapid acquisition of a foreign language by the avoidance of speaking. *International Review of Applied Linguistics* 11:295-317.