ERROR-ADAPTIVE COMPUTER-ASSISTED LANGUAGE LEARNING FOR GERMAN

by

Gertrud Heift

I. Staatsexamen, Weingarten University, 1983
II. Staatsexamen, Weingarten University, 1985

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ARTS

in the Department

of

Linguistics

© Gertrud D. M. Heift 1993

SIMON FRASER UNIVERSITY

September 1993

All rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without permission of the author.

Approval

Name: Gertrud Heift

Degree: Master of Arts

Title of Thesis: Error-Adaptive Computer-Assisted Language Learning for German

Examining Committee:

Chair: R. DeArmond

Dr. Hector Hammerly
Senior Supervisor
Professor of Applied Linguistics

Dr. Paul McFetridge
Professor of Linguistics

Dr. Juan Sosa
External Examiner
Professor of Spanish/LAS

Date Approved: September 3, 1993

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Error - Adaptive CALL for German

Author: _____
(signature)

GERTRUD HEIFT
(name)

July 29, 93
(date)

Abstract

The author presents an Intelligent Tutoring System for German language instruction and outlines the theoretical rationale underlying its design. The examplar is subordinate clauses, a classic problem in German pedagogy. The system is implemented in HyperCard v.2.1, running on Macintosh System 7.1.

A review of the literature provides historical and pedagogical perspectives on computers and computer-assisted language learning (CALL). The author contrasts and compares three teaching approaches, behaviorist-structural, explicit learning, and acquisitionist, examining their application to CALL systems. The eclectic teaching method employed in the software draws upon all three teaching theories.

The reader is introduced to various modes of computer instruction, and an argument for Intelligent Tutoring Systems is presented. Designed as an Intelligent Tutoring System, the software leads the student through a graded, individualized, or, *individually tailored* , program.

The interactive model strikes a balance between student and computer control, allowing for user-initiated actions limited only by pedagogical considerations. Informative interaction provides the user with error-contingent feedback, that is, feedback which not only signals student error but also targets specific errors, taking appropriate action, and directing the student to further exercises. The program implementation simulates intelligence through the use of daemons: conceptually simple, modular sub-programs dedicated to

specific errors. Overall design is consistent with the extant literature on computer error message and dialog design, specially adapted by the author to the specific needs of the language learner.

In conclusion, intelligent responses to student's input can be simulated on a micro computer platform. The strategy described exploits the constrained and predictable domain of the language exercise to efficiently achieve the teaching goals.

Acknowledgements

Heartfelt thanks to Professors H. Hammerly and P. McFetridge for their insights and assistance throughout this project. And very special thanks to Chris.

CHAPTER 1 Introduction

The intent of this project is to outline an Intelligent Tutoring System which can be used efficiently and effectively in the second language classroom. Language classes, particularly those for academic and professional purposes, primarily concentrate on the teaching of form, leaving little time for communicative activities; that is, most of the teaching time is spent on the rather mechanical practice of the various grammatical constructions of the target language. In addition, large class sizes prohibit extensive individualization of the learning process. Communicative activities and individualization are, nonetheless, important. The former allow students to spontaneously apply the rules learned during more constrained exercises; the latter permits students to work at their own pace. In practice, the momentum of the group must often take precedence over the need to correct individuals' errors. The dilemma, then, for the language teacher is to find sufficient time for conversation, role playing, etc., while still pursuing the grammatical essentials.

The software accompanying this project acts as a computer tutor providing the student with efficient and effective practice in one of the formal aspects of German, subordinate clauses. Subordinate clauses are used only by way of example: the modular design of the system is inherently flexible and can be easily expanded to incorporate additional formal aspects of German. A language teacher employing this software can better manage classroom time, relying on the computer to provide efficient and individually tailored practice of grammatical structures and thus freeing more time for teacher-led communicative activities.

CHAPTER 2

History and Teaching Methodology of CALL

2.1 Historical Development of CALL

Modern Computer-Assisted Language Learning (CALL) is the result of the convergence of several fields of research addressing the use of computers in language processing. While the influence of computational linguistics and machine translation may be indirect, CALL systems draw heavily upon the findings of both areas. The handling of natural language by the computer essentially contributes to the fluency of interaction between the human user and the machine. Ahmad (1985) gives two reasons for the importance of computational linguistics and machine translation to CALL:

> "First, those working in these areas have provided some of the tools for CALL, such as particular features of programming languages; second, they form part of the groundwork for future developments, as they will lead to

> more intelligent processing of grammar and meaning and therefore to more sensible responses from the computer."[1]

We are, ultimately, aiming at interactive computer systems possessing a high degree of artificial intelligence fully capable of processing natural language input. This represents an even more complex task in the educational, as opposed to the commercial or military, field since natural language constitutes not only the operational dialogue, but the subject matter itself.

We find a direct bearing on CALL in experiments in Programmed Instruction (section 2.1.1). In the late sixties CALL systems were being developed mostly on large-scale, mainframe systems in universities where computer sessions were intended to replace classroom instruction. Of course, as Holmes and Kidd (1982) point out, much of the structure of the computer programs was created by psychologists and the computer industry itself, rather than by language teachers. Two important projects at that time were the Stanford Project (Stanford University) and the PLATO system (Programmed Logic for Automated Teaching Operations) developed at the University of Illinois and by Control Data Corporation (section 2.1.2).

2.1.1 Programmed Instruction

Throughout the 1960's, Programmed Instruction (PI), which arose out of Skinner's work at Harvard University, dominated the field of computer-assisted learning. Even in the early 80's it was still assumed that computer-assisted learning and Programmed Instruction

1. Ahmad, K., Computers, Language Learning and Language Teaching, p. 28.

were essentially one and the same; the microcomputer was merely the latest technical device to bring the behaviorist view of language teaching back into the classroom (Kaliski 1992). Programmed Instruction proposed that the best way of learning a task is

> "to break it down into a series of small subtasks and then tackle each one of these in turn. As a means of reinforcing learning, student mastery of a subtask would be rewarded in an appropriate way."[2]

The generally sound pedagogical principle of dividing a large learning task into conceptually smaller units, was, however, distorted by an over-emphasis on the reinforcement aspect of Skinner's stimulus-response-reinforcement paradigm. The result was automated teaching machines which presented the student with 'mindless'[3] drills, typically multiple choice answers, and provided inappropriate feedback:

> "... a student was presented with a problem and then asked to type in the answer through a typewriter-like console. The answer was checked as each character was entered. If any error resulted, the computer assumed control and typed out the word 'wrong'. An incorrect answer then caused the generation of another problem of similar difficulty. After a number of successive incorrect answers a problem of lesser difficulty would be generated."[4]

Beyond any purely humanitarian concerns with Skinner's psychology, we have to question the effectiveness of such systems. A program with a simple letter to letter match is incapable of differentiating types of errors: not only is it, therefore, incapable of

---

2. Barker, P. &Yeates, H., Introducing Computer Assisted Learning, p. 45.
3. While there are many kinds of drills, 'mindless' refers here to simple substitution/ pattern drills.
4. Barker, P. &Yeates, H., Introducing Computer Assisted Learning, p. 45.

providing any valuable, evaluative feedback, but, in ignoring the source of the error when selecting another problem, it would seem to be relying upon an inflexible, program-centered, rather than student-centered, definition of difficulty. While, in considering the early systems, we should make allowances for the limitations of the then current technology, we must also note that, although PI has been redefined over the years, it has never achieved a high degree of popularity (Price 1991). The original linear programs (representing fixed sequences of instructions) improved, becoming more sophisticated branching programs; most, however, remained based upon multiple choice answers. Inasmuch as the new is often an offshoot of the old, as among teaching approaches which emulate some procedures of the previous approach while rejecting others, CALL is historically related to PI, adopting many of its concerns. As Price states:

> "... concern for individualized learning, self-pacing, immediate knowledge of the correctness of response, and reinforcement are just as important today as they were to PI twenty five years ago."[5]

2.1.2 The Stanford Project and PLATO System

The Stanford Project which was carried out under the supervision of J. Van Campen in the Slavic Languages Department was a self-instructional introductory course to Russian. While the student typed answers to questions, inflected words, and performed various types of transformation exercises on the computer, a tape-recorder connected to a teletype was used solely to ask questions. The system

5. Price, R.V., Computer-Aided Instruction, p. 12.

did not allow for the recording of answers; recordings of the student output were produced in the traditional language laboratory. While all of the student's work was evaluated by a language teacher, the computer clearly superseded the teacher in the actual instruction. It is important at this point to relate this project to the general language teaching philosophy of the time. Van Campen states that

> "the introduction of computer-based instruction in elementary language courses in which the acquisition of writing plays an important role would greatly improve the effectiveness of those courses."[6]

The influence of Audiolingualism, a behaviorist-structuralist approach to language teaching, becomes evident. Among some of the audiolingual methodologists there is a focus on teaching all four language skills, listening-speaking-reading-writing, at the same time and with equal emphasis. However, as Hammerly (1986) points out in his balanced approach to second language learning, "early access to a second language through its written form --if it has an alphabet-- guarantees the formation of bad pronunciation habits."[7] In addition, although these early computers were not equipped with sound, we can see the influence of Audiolingualism in the students' being presented with exercises/drills in a language laboratory focussing on aural/oral skills.

A general concern among teachers of the period was that the computer would alter the student/teacher relationship (Kaliski 1992). The Stanford Project went as far as to replace the teacher all but

6. Van Campen, J., "A Computer-Based Language Instruction System with Initial Application to Armenian", p. 27, as cited in Ahmad, K., Computers, Language Learning and Language Teaching, p. 29.
7. Hammerly, H., Synthesis in Language Teaching, p. 188.

entirely, while "at New York the teachers working with computers in the language teaching programs were reduced to the role of teaching audiolingual drills."[8]

The PLATO program, by contrast, was not as dedicated to making the teacher redundant. The original program provided the student with software to practice reading and translation skills from Russian to English. The whole course was divided into three parts: vocabulary skills, grammar explanations and drills, and translation tests. PLATO presently covers a wide range of languages, as well as other subjects taught at the university. Some programs, like the one dedicated to Spanish, are designed as self-instructional software, while others are intended only to complement classroom instruction. Over the years, the system has incorporated new advances in technology, such as the touchscreen and multimedia (e.g. audio in/ output), and even provides the teacher with an authoring language (Wyatt 1984). But, as Ahmad points out "while PLATO offers a high degree of technical sophistication, it is also extremely expensive."[9] The main reason for the cost is that the PLATO software can only be run on specialized hardware consisting of two mainframe computers.

With the advent of the microchip and microcomputers, computer literacy increased dramatically. The computer became available at increasingly affordable prices to individuals as well as to institutions. School Districts/Boards throughout the U.S. and Canada made large commitments to computer education by purchasing

8. Holmes, G. & Kidd, M.E., "Second Language Learning and Computers", Canadian Modern Language Review, vol. 38, p. 504.
9. Ahmad, K., Computers, Language Learning and Language Teaching, p. 32.

computers and software for classroom use. The availability of the hardware also offered programming opportunities to other than professional programmers. In addition, with the movement toward individualized instruction, in the sense that the student works at his/her own pace, the computer began to be seen as a useful tool in the learner-centered classroom. Whereas the original rejection of behaviorism implied a rejection of CALL as well:

> "[M]ethodological changes to a more communicative approach led to a rejection by some teachers of any method associated with behaviorism. It was argued that language with no context, as was the case with many CALL exercises, was of no use and teaching should concentrate on the functional or notional uses of language."[10]

with computers becoming more accessible and accepted, teachers' concerns over being replaced by computers abated, shifting toward "becoming computer literate and incorporating CALL into the curriculum... [as] a matter of professional survival"[11]. Rather than being closely associated with one stream of pedagogic thought, software design in CALL began to reflect more divergent teaching approaches.

10. Kaliski T., "Computer-Assisted Language Learning (CALL)". In Roach P. (Ed.), Computing in Linguistics and Phonetics, p. 99.
11. Teichert, H. U., Computer-Assisted Instruction in Beginning College German: An Experiment, p. 18.

2.2 Teaching Methodology

In the same way teaching philosophies influence the development of printed language learning materials, computer software also reflects the goals of the particular teaching approach. Is the purpose of language merely to fluently communicate or is our emphasis on fluent and accurate communication? When writing or evaluating computer software this question has to be addressed.

2.2.1 Language Teaching Approaches

In relating computer software to language teaching approaches it is important to distinguish between the computer as a medium and the software itself. While the computer medium is not tied to any specific language teaching approach or method, that is, the computer can be used by a vast range of teaching methodologies, the specific language learning software reflects more directly the language teaching pedagogy adopted.

Throughout the history of CALL we find essentially three approaches to language teaching:

1. Behaviorist/Structuralist Approaches
2. Explicit Learning Approaches
3. Acquisitionist Approaches

While many teaching methods, such as the Structural Method which combined aspects of both the Behaviorist/Structuralist and Explicit Learning Approaches, incorporated theoretical underpinnings from more than one approach, the following discussion will examine

each in rather stark isolation to better understand their implications in CALL.

The behaviorist/structural approach to language learning is based on behaviorist learning theories developed by Skinner and the structuralist linguistic theory founded by Bloomfield. The manifestation of these ideas is found in Audiolingualism, a method based on psychological and linguistic theory. Language learning is seen as a habit formation through stimulus, response, and reinforcement. Language teaching employs a bottom-up approach: language is split into small units, where the successful completion of one building-block leads to the next one. The key to this approach, as manifested in its later distorted, extreme forms, is 'mindless' drilling and rote memorization: grammar is taught inductively with little rule explanation and formulation. In addition, in this habit-formation theory, situations where frequent errors are produced are to be avoided, since they could lead to the formation of bad habits. It was this reasoning which led to a complete control of the pattern drills.

For CALL software to be representative of this approach Hubbard (1987) lists a number of principles:

1. presents vocabulary and structure appropriate to the learner's level
2. maintains the learner's attention to task
3. does not accept errors as correct answer
4. requires the learner to input the correct answer before proceeding
5. provides the learner with positive feedback for correct answers
6. provides sufficient material for mastery and overlearning to occur

7. reinforces patterns and vocabulary presented in a lesson
8. presents grammar rules or patterns inductively with no attempt at teaching explicit formulations of them[12]

In contrast to the behaviorist/structural theory we find the explicit learning approaches, as manifested in the Cognitive Approach. This approach is based on cognitive learning theories, that is, the student is supposed to be cognitively involved in the learning process. Furthermore it reflects the theory of transformational-generative grammar proposed by Chomsky. Language is seen as rule-governed creativity, using rules to create meaningful[13] utterances. Explicit learning approaches are structurally graded where grammar rules are taught deductively, and to be consciously acquired. Therefore

> "language is not a set of habits to be mindlessly drilled, but the creative use of internalized rules. These rules are complex and abstract and do not lend themselves easily to conscious formulation, but constitute instead our unconscious "competence" which makes it possible to generate an infinite array of new sentences."[14]

In transformational-generative theory errors are seen as testing hypotheses of the concepts students form in the learning process. Their purpose is to provide the teacher with cues of the student's competence level.

---

12. Hubbard, P. L., "Language Teaching Approaches, the Evaluation of CALL Software, and Design Implications". In Smith, Wm. Flint (Ed.), Modern Media in Foreign Language Education: Theory and Implementation, pp. 231-2.
    Hubbard's eighth principle seems to be more a reflection of the Naturalistic Approaches than of the Behaviorist/Structuralist. Only one version of the Audiolingual Methods -- that is, a variety close to the Direct Method showed reluctance toward rule formulation/explanation.
13. "Providing drill-and-practice material in explicit learning approaches that is meaningful, contextualized, and interesting to the students is a recent trend that parallels developments in communicative approaches."Hubbard, P. L., "Language Teaching Approaches, the Evaluation of CALL Software, and Design Implications". In Smith, Wm. Flint (Ed.), Modern Media in Foreign Language Education: Theory and Implementation, p. 233.
14. Underwood, J. H., Linguistics, Computers and the Language Teacher, p. 11.

CALL software will be representative of explicit learning approaches if it does the following:

1. introduces or reviews grammar rules and word meanings in an understandable, learnable, and reasonably accurate form[15]
2. provides effective practice so that (a) novel target-language input can be readily understood, and (b) the learner's understanding of rules leads to the production of grammaticality acceptable spoken or written target-language discourse in novel situations
3. gives meaningful rather than mechanical practice
4. gives practice contextualized in a coherent discourse larger than a single sentence
5. provides hints of various types to help lead students to acceptable answers
6. accepts alternative correct answers within the given context
7. provides the student with explanation of correct answers
8. anticipates incorrect or inappropriate answers and explains why such answers are incorrect and inappropriate
9. maintains the student's interest throughout the exercise
10. allows an appropriate degree of student control[16]

Even more removed from the behaviorist/structuralist theory, we find the acquisitionist approaches to language learning, as manifested first in the Natural Approach. This theory, developed by Stephen Krashen in his *Monitor Model*, emphasizes that a second language is acquired in an unconscious and natural way similar to

15. Hubbard's description of the grammar approach used in CALL seems to contradict the underlying notion of a descriptive grammar which by definition is scientific and not easily understandable for second language learners.

16. Hubbard, P. L., "Language Teaching Approaches, the Evaluation of CALL Software, and Design Implications". In Smith, Wm. Flint (Ed.), Modern Media in Foreign Language Education: Theory and Implementation, pp. 233-4.

first language acquisition. The key to this model is to provide the student with comprehensible input just slightly higher than the stage s/he is at (i+1). The input has to be relevant and interesting to the student, an idea shared by the notional/functional syllabus. It is a top-down approach where material is presented as in conversation: no special attention is drawn to new items, drilling, or grammar. The focus is on meaning and not on form. Errors remain uncorrected since, in doing so, the emphasis on meaning would shift towards form. Accuracy is supposed to be acquired as a by-product of fluency.

CALL software representative of this approach:

1. provides meaningful communicative interaction between the learner and the computer
2. provides comprehensible input at a level just beyond that currently acquired by the learner
3. promotes a positive self-image in the learner
4. motivates the learner to use the software
5. motivates the learner to learn the language
6. provides a challenge but does not produce frustration or anxiety
7. does not include overt error correction
8. allows the learner the opportunity to produce comprehensible output
9. acts effectively as a catalyst to promote learner-learner interaction in the target language[17]

As with regular classroom instruction, we find a trend towards acquisitionist approaches in computer-assisted language learning. Kaliski (1992) states that

"the type of program, which appears to be the biggest

17. Hubbard, P. L., "Language Teaching Approaches, the Evaluation of CALL Software, and Design Implications". In Smith, Wm. Flint (Ed.), Modern Media in Foreign Language Education: Theory and Implementation, p. 236.

> selling and the most popular with students, comes in the 'computer as a playmate' category. These are word guessing games which use variations of the conventional cloze techniques or jumbled sentences."[18]

These kinds of computer programs are, at least initially, highly motivating for the student, a not unimportant consideration. There is, however, much debate in the field of language teaching over the ultimate effectiveness of this pedagogy.

A more practical and disturbing problem in CALL is that

> "current CALL programs [have no relationship to theories of language and] often appear to have been designed merely to demonstrate technological developments rather than contribute to the educational process." [brackets added][19]

While the above statement calls for our attention to the problematic underemphasis of language teaching pedagogy in CALL, it is not quite right to say that current programs do not reflect any teaching approach. All educational software design implicitly incorporates one, or more likely many, different approaches, albeit in a haphazard or ad hoc fashion. There may also be further distortion in the implementation of an idea borrowed from a teaching pedagogy owing to design costs, hardware limitations, and/or the interdisciplinary nature of the task at hand, this latter meaning that programmers are not usually language teachers and vice versa. Nonetheless, Hubbard argues that most CALL features can be traced to one of three teaching approaches. *Table 2-1: CALL Features* makes

---

18. Kaliski T., "Computer-Assisted Language Learning (CALL)". In Roach P. (Ed.), Computing in Linguistics and Phonetics, p. 99.

19. Kaliski T., "Computer-Assisted Language Learning (CALL)". In Roach P. (Ed.) Computing in Linguistics and Phonetics, p. 99.

explicit those features and their theoretical parentage. In some cases, of course, the particular CALL feature may only roughly correspond to its classroom equivalent; nonetheless, from a design point of view, orienting potential features within the larger theoretical framework will help us attain a more coherent overall design.

Features	Behaviorist/ Structural Approaches	Explicit Learning Approaches	Acquisitio-nist Approaches
Emphasis	on form	on form but with awareness of meaning	on meaning
Form of Exercises	multiple choice, pattern drills	exercise material is meaningful and presented in context	communicative, mostly in form of games
Presentation of Grammar	graded, descriptive grammar	graded, descriptive grammar	ungraded, grammar is not taught overtly
Errors	deterred but corrected	corrected	uncorrected
Progressive Error Correction	no	yes	- - -
Error Feedback	"Wrong, try again!" (or the correct answer is provided)	students are led toward a correct answer, provides explanation for correct answer •	- - -
Motivation	not explicitly considered	considered	considered
Control	computer	student/computer	student

Table 2-1: CALL Features

Analyzing the similarities and differences of these three theories, we note that the explicit learning approaches are very nearly a combination of the principles of the others. Teaching occurs in a student-centered classroom in which, however, an emphasis on form is maintained through the presentation of graded material. The only unique feature, that is, the only feature not contained in any of the other approaches, is the progressive error correction for incorrect or inappropriate student responses, an area closely related to error feedback. In the cognitive learning approaches errors are corrected by leading the students toward the correct answer. This feature does not apply in the acquisitionist approaches, since grammar is not explicitly taught and incorrect responses are not intercepted by the system. In the behaviorist theory, grammar is often taught inductively -- hardly any explanations/rule formulations are given. If errors do occur, however, they are corrected in a far less elaborate way, either by providing the student with the correct answer or simply signalling that an error occurred.

While taking into account aspects of all three approaches, the computer program associated with this project does not strictly adhere to any one of them, incorporating, rather, elements of all three into its teaching methodology. The choice of a more eclectic method is another common trend in teaching. While language approaches have often developed in reaction to previous schools of thought, for example, Audiolingualism's rejection of the Grammar-Translation Method, we nowadays find teachers extracting principles from a variety of methods and approaches. Consider the principles implemented by the proposed model in *Table 2-2: CALL Features:*

Features	Proposed Model
Emphasis	on form but with awareness of meaning
Form of Exercises	tutor, game
Presentation of Grammar	graded, pedagogical grammar
Errors	corrected
Progressive Error Correction	yes
Error Feedback	students are led toward a correct answer
Motivation	considered
Control	student/computer

Table 2-2: CALL Features

We find close similarities with the explicit learning and acquisitionist theories with regards to the student role in the learning process. The approach taken is that learning can only take place if the "Affective Filter", a notion proposed by Krashen (1977), is low. Motivation, self-confidence, and anxiety are central factors in learning. They reflect the learners' attitudes and determine how the information gets "filtered". If the attitudes are negative, new information is strongly filtered, meaning that learning is hindered. Closely connected to this concept is the question of who is in control of the learning process; the extreme positions are staked out by the behaviorist/structuralist and the acquisitionist theories. The approach taken in this project presupposes that the teacher's role is to guide the student through the learning process by presenting graded material, but reserving some creativity/control for the student. The key idea here is that pedagogical concerns should play just as

important a role in software design as in regular classroom instruction.

As opposed to a scientific grammar a *pedagogical* grammar has been chosen. The use of pedagogical grammars dates back to the Structural Method, an early form of the behaviorist/structuralist approaches. It seems rather obvious that, if we are teaching grammar, we need a grammar designed for teaching purposes. A pedagogical grammar is

> "a grammar designed to take a learner where he is (taking into account his knowledge of the native language and his traditional and popular notions about language) and lead him step by step to the internalization of the structure of a second language,..."[20]

The design of a step by step program, where the emphasis is placed on form with awareness of meaning, presents the student with graded material. The student is provided with explanations of incorrect responses, a *deep* error correction, to lead him/her toward a correct answer. The rationale behind this decision is that rule internalization, a precondition for creative use of language, can only occur if cognitive involvement on the part of the student is ensured. This particular kind of error feedback is achieved by having the computer act as an Intelligent Tutoring System. The various modes of the computer and the functions of tutoring systems, in particular, will be discussed in Chapter 3.

20. Hammerly H., An Integrated Theory of Language Teaching, p. 124.

CHAPTER 3

Teaching Subordinate Clauses in German with CALL

3.1 Teaching Functions of the Computer

Price (1990) lists five distinct 'modes' of computer instruction:

1. *Page Turners*
2. *Simulations*
3. *Educational Games*
4. *Drill and Practice*
5. *Tutorial*

While each mode has its uses, from on-line manuals to expert-systems, not all are equally suitable for CALL.

Page Turners use the computer screen as a *page* to display the knowledge to be transmitted. Although similar in this respect to any printed material, they have the added advantage of potentially

incorporating animation, sound, and other media. While the "living textbook"[1] may be well fitted to representing spatial concepts and time-dependent trends, or navigating large data-bases, studies[2] have shown that people assimilate purely text-based material more quickly, and with less effort, from traditional print sources. Even the more elaborate multi-media versions have only limited application in CALL: the user's passive role provides no opportunity for the practice so essential to language learning.

Simulations allow students to deduce general concepts by responding to ostensible situations. The computer lets the student explore topics, usually through animation, rather than leading him/her toward explicit outcomes. These programs are not only highly motivating for students but allow them to encounter a wide variety of situations *virtually*, usually at far lower cost, and with less dire consequences in the event of student error. Due to the expense of constructing a convincing virtual reality, the most elaborate simulators to date have been contracted for and built by the military, for use as flight-trainers, and battle-simulators; more recently, however, we find simulators being used in the medical, engineering, and architectural professions. In CALL, simulations find some applicability, inasmuch as they can offer a wide range of exposure to the second language culture.

Educational Games provide the student with drill and practice embedded in a game to make it more memorable and motivating. An example might be using a bingo game to practice vocabulary. Such

1. Price, R.V., Computer-Aided Instruction - A Guide for Authors, p. 32.
2. Shneiderman, B., Designing the User Interface, p. 362.

games are simple to encode and, as a result, inexpensive. However, in the second language classroom, Educational Games require some prior subject knowledge, presumably gained from a human teacher.

Drill and Practice programs are constructed without game elements to provide the student with activities intended only to reinforce classroom instruction. "Problems are generated by the computer program whereas in conventional methods drill problems are found at the end of textbook chapters, on worksheets, or on flashcards."[3] The computer is, in this case, merely the latest device in teaching, offering no overt advantages over textbook chapters or workbooks. *Authoring programs*, which allow teachers to create their own exercises tailored to particular students' needs, are superior to fixed drills, but offer no especial advantage over worksheets.

Tutorials, in contrast, do not rely on previous instruction. Rather they assume the role of a human tutor, guiding the student toward a given task. They generally offer some degree of diagnostic testing, more corrective feedback than a simulator or a game, and more flexible branching within the program than found in Drill and Practice software. The simpler implementations bear a striking resemblance to earlier Programmed Instruction systems, while sophisticated examples emulate the human tutor to a modest degree. As will be discussed in section 3.2, however, significant aspects of the teacher/student interaction are still missing.

3. Price, R.V., Computer-Aided Instruction - A Guide for Authors, p. 27.

3.2 Intelligent Tutoring Systems

The foregoing classifications provide a general framework for the different approaches to computer instruction, but do not address the communication between user and computer, an integral part of the learning environment. If we are aiming at optimal instruction, one of the most important issues is the individualization of the learning process.

> "The major advantage by the computer is individualization. In the typical foreign language class, students vary widely in their instructional needs, and research shows that students learn a foreign language effectively if they spend useful instructional time on tasks suited to their own needs. The teacher is faced with major difficulties because in a class of thirty students, some will have mastered all the prerequisite skills to begin working toward mastery of a designated objective, while others will have mastered none of them."[4]

But the teacher/student ratio can only effectively be reduced using computers if we provide the student with a device capable of realistically approximating the teacher/student interaction.

Increasingly, more attention is being paid to the interactive aspects of tutoring systems[5]. An optimal system must simulate a high degree of intelligence. An Intelligent Tutoring System is a

> "computer program that

4. LaReau, P., Vockell, E., The Computer in the Foreign Language Curriculum, p. 72.
5. "So in the 1990s our educational and training establishments are witnessing evolutionary advances in intelligent computer-assisted instruction and the emergence in the research laboratories of intelligent computer systems (ITSs)." Burns, H., Parlett, J.W, Redfield, C.L. Intelligent Tutoring Systems, p. xi.

a. is capable of competent problem solving in a domain
b. can infer a learner's approximation of competence, and
c. is able to reduce the difference between its competence and the student's through application of various strategies."[6]

Designers of Intelligent Tutoring Systems are not only concerned with the subject matter, but also with the interaction between the student and the computer, so that the primary goal of learning can be achieved more efficiently.

> "Communication knowledge in an ITS consists of rules and facts that tell the system how to manage the student-computer interaction. ... Thus communication knowledge refers to the conversation between the student and the computer system and all the media necessary for that communication. Together these elements comprise what may be called *communication style*."[7]

Categorizing programs according to communicational styles organizes them along lines not dissimilar to Price's classification of Page Turner, Simulations, etc., but with an important new emphasis upon user/computer interaction. Fischer and Morch (1988)[8] describe three different communication styles in Intelligent Tutoring Systems:

1. *Consultation*
2. *Critiquing*
3. *Tutoring*

In the Consultation Model, as implemented in expert systems, for example, the user is able to ask questions of the computer

6. Wenger, E., Artificial Intelligence and Tutoring Systems, p. 263.
7. Burns, H., Parlett, J.W., Redfield, C.L., Intelligent Tutoring Systems, pp. 17-8.
8. Fischer, G., & Morch, A. "Crack: A Critiquing Approach to Cooperative Kitchen Design". In Proceedings on Intelligent Tutoring Systems, pp. 176-185. New York: Association for Computing Machinery, as cited in Burns, H., Parlett, J.W., Redfield, C.L., Intelligent Tutoring Systems, pp. 17-8.

consultant. An example would be a medical diagnostic system where a physician enters the medical history, vital statistics, and symptoms of a patient in order to obtain a recommendation for testing and treatment. The interface of these systems must take into account the knowledge representations of the users, and be able to query and respond in a pseudo-intelligent manner.

In Critiquing Models, the student acquires problem-solving skills through exploring concepts and testing hypotheses. "The computer interrupts only when the student fails to meet minimum criteria. ... [It] allows the student to fail and make mistakes."[9] Students pursue their own goals through multiple paths which they can access in a non-sequential order. Simulators, discussed earlier, would tend to fall under the Critiquing Model, inasmuch as the trainee is free to take many actions until such time as the plane 'crashes'.

The Tutoring Model, in contrast, while student-centered in an educational sense, does not rely as heavily upon an event-oriented, user-centered interface. The program leads the student through a sequence of instructions, where the arrangement of the paths are determined by the designer. A high degree of sophistication in the areas of corrective feedback, diagnostic testing, and multiple pathways, means that branching decisions are primarily handled by the program, thereby ensuring the attainment of the teaching goal. Tutoring models are very suitable for subjects dealing with formal rules because they provide students with graded, effective practice, allowing them to internalize concepts. For these reasons the program

9. Burns, H., Parlett, J.W, Redfield, C.L., Intelligent Tutoring Systems, p. 19.

accompanying this project adheres largely to such a model. It should be recalled that in the design of Intelligent Tutoring Systems consideration is given to both:

1. the computer interaction, and
2. the instructional goal.

An event-centered design has been retained as much as is compatible with instructional goals, therefore, providing a balance of student/computer control. The interactive model employed will be discussed at fuller length in Chapter 4.

The instructional goal is achieved by integrating teaching principles, such as gradation and individualization, into a branching program. The teaching principles and the overall architecture of the program will be discussed in section 3.3.2.

3.3 Overview and Teaching Objectives

3.3.1 Subordinate Clauses

Subordinate clauses, a classic problem in German pedagogy, is taught at the late beginner's level of second language instruction. The position of the verb in a German sentence depends on whether it appears in a main clause or subordinate clause. Whereas a German main clause has an SVO word order[10], a subordinate clause has an SOV word order. The following example shows that the verb 'liest' appears in second position when in a main clause, but sentence-final when in a subordinate clause.

Main Clause:

S	V	O
Sie	liest	ein Buch.
She	is reading	a book.

She is reading a book.

Subordinate Clause:

		S	O	V
Ich weiß,	daß	sie	ein Buch	liest.
I	know that	she	a book	is reading.

I know that she is reading a book.

3.3.2 Program Architecture and Teaching Principles

Inherently, the use of a pedagogical grammar in a computer-instruction program means presenting grammatical constructs gradually. Gradation implies a carefully selected sequence of material, proceeding "from simple to complex, the frequent before the infrequent, the concrete before the abstract, the independent before the concomitant, etc."[11]

In addition, if our goal is to individualize the learning process through use of an intelligent tutor, we need a program with branching capabilities. Branching programs depart from the usual sequence of executing instructions in a computer in order to address the particular deficiencies of a student.

A computer, with its branching abilities, can reduce the

10. This already by itself causes problems to English speakers learning German since we find the verb in a German main clause always in second position.
E.g.: Heute gehe ich ins Kino.
Today go I to the movies.
Today I am going to the movies.

11. Hammerly, H., An Integrated Theory of Language Teaching, p. 117.

> repetition for those who do not need additional practice. For students who do need the extra practice, creative teachers overcome this difficulty ..., by supplying informative feedback, and by making the repeated practice relevant to the present or future needs of the students."[12]

Consider Figure 3-1:

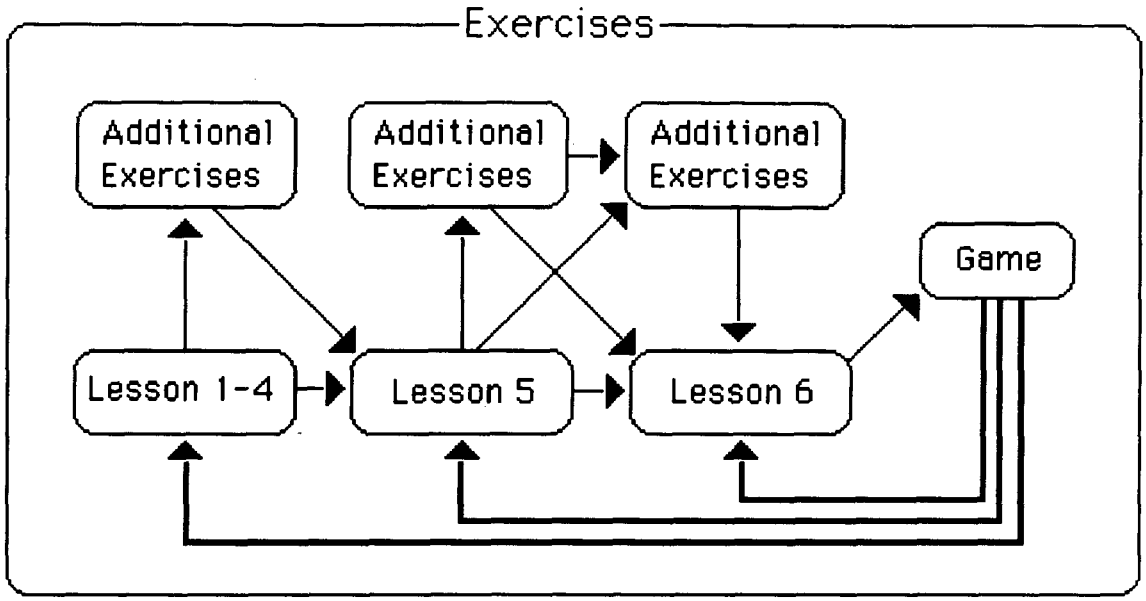

Figure 3-1

The program is organized into three major sections and a final game. Each student takes an individual path depending on his or her mastery of each subtask. Cumulatively, these tasks lead to an understanding of subordinate clauses in German.

The objectives in Lesson 1-4 are:

1. correct use of subordinating conjunctions
2. correct word order

12. LaReau, P., Vockell, E., The Computer in the Foreign Language Curriculum, p. 72.

3. correct punctuation (the comma precedes the subordinating conjunction)
4. correct spelling

The objectives are sought as follows:

- Lesson 1: In Lesson 1 the students are presented with a paragraph where the subordinating conjunctions are missing. The student's task is to provide the correct subordinating conjunction in context.

Example exercises from Lesson 2-4 are:

- Lesson 2: Join the sentences[13]:

  Ich nehme zwei Menüs. Ich habe Hunger. (because)

  Ich nehme zwei Menüs, **weil** ich Hunger **habe**.
- Lesson 3: Join the sentences:

  Sie fragt, ...

  Wo wohnt Peter?

  Sie fragt, **wo** Peter **wohnt**.
- Lesson 4: Begin the sentence with the subordinate clause:

  Er hat eine Party gegeben, weil sie 21 geworden ist.

  Weil sie 21 geworden **ist, hat** er eine Party **gegeben**.

After Lesson 4 the student's performance is evaluated[14]. Branching provides the student with additional exercises as needed to ensure his/her mastery of each grammatical step. The additional exercises accompanying Lesson 1-4 are similar in make-up to the examples above. Should the student require remedial practices, s/he goes through the additional exercises, and then, returning to the main path, continues on to Lesson 5.

13. The instructions are only given in English with the first exercise in each Lesson.
14. The way the system handles students' evaluations and error thresholds will be discussed in detail in Chapter 4.

In Lesson 5 the task grows more complex: the student must begin to correctly handle verb inflection, gender and Case of nouns.

The objectives for Lesson 5 are:

1. correct word order
2. correct punctuation
3. correct verb inflection
4. correct use of gender and Case of nouns
5. correct spelling

Consider the following example for Lesson 5:

- Lesson 5: Form a sentence:
  Er / fragen / ob / Salat (def. article) / schmecken / gut.
  Er **fragt, ob der** Salat gut **schmeckt**.

After Lesson 5, there are <u>two</u> sets of possible additional exercises provided: the first, specifically addresses gender and Case of nouns, and the second, practices word order in subordinating conjunctions. Both sets continue to address verb inflection. The additional exercises have been separated in order to correct any deficiencies in gender and Case of nouns, before addressing word order. The first set of additional exercises on gender and Case presents only a main clause, containing two noun phrases. The second set covers word order in subordinate clauses, and incidentally, gender and Case.

Whereas up to this point, all the German words are in one way or another provided, Lesson 6 requires the student to translate sentences, while applying all of the concepts practiced so far.

The objectives for Lesson 6 are:

1. correct translations

2. correct spelling
3. correct word order
4. correct use of subordinating conjunctions
5. correct verb inflection
6. correct use of gender and Case of noun
7. correct punctuation

After Lesson 6, the student moves on to the final game after which s/he can start again with one of the sets of exercises.

In the final game, the student is presented with six sentences, one at a time, in random order; each of the sentences is missing the subordinating conjunction. The student is also provided with six buttons, each labelled with a subordinating conjunction. The challenge is to provide the correct answer as quickly as possible with a timer at the edge of the screen keeping score. The game is intended to be fun and to reinforce the grammatical concepts through a different mode of computer instruction.

CHAPTER 4

Human Factors in CALL

4.1 Interaction in CALL

Marshall (1988) identifies the "significant interactive qualities"[1] of computer assisted language learning as one advantage of introducing the computer into the classroom. True interaction requires intelligent behavior on the part of the computer as well as the student. Without intelligence, defined here as *informative interaction*, the computer is merely another medium for presenting information, one not especially preferable to a static medium such as print. In order to go beyond multiple choice questions, relatively uninformative answer keys, and gross mainstreaming of students characteristic of workbooks, the proposed model emulates significant aspects of a student-teacher interaction.

1. Marshall, D. V., CAL/CBT - The Great Debate, p. 17.

Section 4.2 presents an example of how the system handles errors of gender versus Case[2] in German providing evaluative feedback. In addition, Section 4.3 outlines error message guidelines which take *motivation*, as a central aspect of learning, into account. Section 4.4 discusses the overall design of the interface as implemented in the system.

4.2 Informative Feedback

In terms of ease of implementation, the simplest way for a program to evaluate a student response is a straightforward string match. That is, the student response is compared letter for letter against an answer key. For a program to be informative, however, it must do *more* than merely indicate that an error has occurred: the software must also give a description of the error, and perhaps go to an even deeper linguistic analysis in order to isolate the source of error. For example, if a student provides a wrong article the error might be either incorrect gender or incorrect Case. In such an instance the program must be capable of distinguishing between the two error types and providing *error-contingent feedback*[3]. While this is quite a complex task in areas involving natural language responses when compared to more quantitative areas, such as mathematics, nonetheless

> "... for almost all cognitive learning, instruction is

2. For clarity grammatical Case is capitalized.
3. "Feedback tailored to the nature of the student's error is called *error-contingent* feedback." Alessi S. M. & Trollip S. R., Computer-Based Instruction, p. 116.

> enhanced by evaluative feedback. In many cases it is essential, if any learning is to occur. Translation of a foreign language is a prime example of the latter situation."[4]

The following code illustrates error-contingent feedback, responding differently to an article which is in the wrong Case than to one which is of the wrong gender. The first subroutine (line 2-9) checks the gender of a noun, the second subroutine (line 10-19) looks for Case.

Code example 4-1 [5]

```
1   On AnswerCheck
2   if word dpos of string is not word det in card field artfield in card artcard then
3   if incorrectInflection(word dpos of string, artfield, artcard) is false then
4   select word RealPosition(punctposition, dpos) in field answer1
5   answer "Wrong gender?"
6   add 1 to first word of card field score in card scorecard
7   exit answercheck
8   end if
9   end if
10  if word dpos of string is not word det in card field artfield in card artcard then
11  if incorrectInflection(word dpos of string, cases, artcard) is false then
12  select word RealPosition(punctposition, dpos) in field answer1
13  answer "Wrong case?"
14  add 1 to second word of card field score in card scorecard
15  exit answercheck
16  end if
17  end if
18  select word RealPosition(punctposition, dpos) in field answer1
19  answer "Wrong translation?"
20  End AnswerCheck
```

4. Venezky, R. & Osin, L., The Intelligent Design of Computer Assisted Instruction, p. 9.
5. The terminology is explained on page 37.

The stack contains one card for the definite articles. This card contains four fields, one for each Case[6], and one which shows all possible articles of the Cases. The Cases concerned are:

Nominative: field 'detnom': der, die, das, die
Accusative: field 'detacc': den, die, das, die
Dative: field 'detdat': dem, der, dem, den.

All possible German articles[7] for the three Cases are:

Field 'cases': der, die, das, den, dem

IncorrectInflection, a function which is present in both of the two subroutines (Code example 4-1, line 2 and 10), checks whether the student's answer is in one of the four fields.

Code example 4-2

```
1   Function IncorrectInflection reply, tense, verbcard
2   set cursor to busy
3   put true into it
4   repeat with increment = 1 to number of words in card field tense in card verbcard
5   if reply is word increment in card field tense in card verbcard then
6   return false
7   end if
8   end repeat
9   return it
10  End IncorrectInflection
```

To follow a concrete example through this process of error analysis, consider the following exercise presented to the student:

Translate the following:
The car belongs to the woman.

6. The 'genitive' is not included since it is not taught until the intermediate level. All articles are listed in the order: masculine, feminine, neuter, plural.
7. For the indefinite articles, or, 'kein', which is considered an 'ein-word', the same concept has been applied and individual fields have been created accordingly.

The correct answer is:

Das Auto gehört der Frau.

A mistake-ridden response might be:

* Der Auto gehört *die Frau.

The following flow-chart (Figure 4-1) illustrates the logical process:

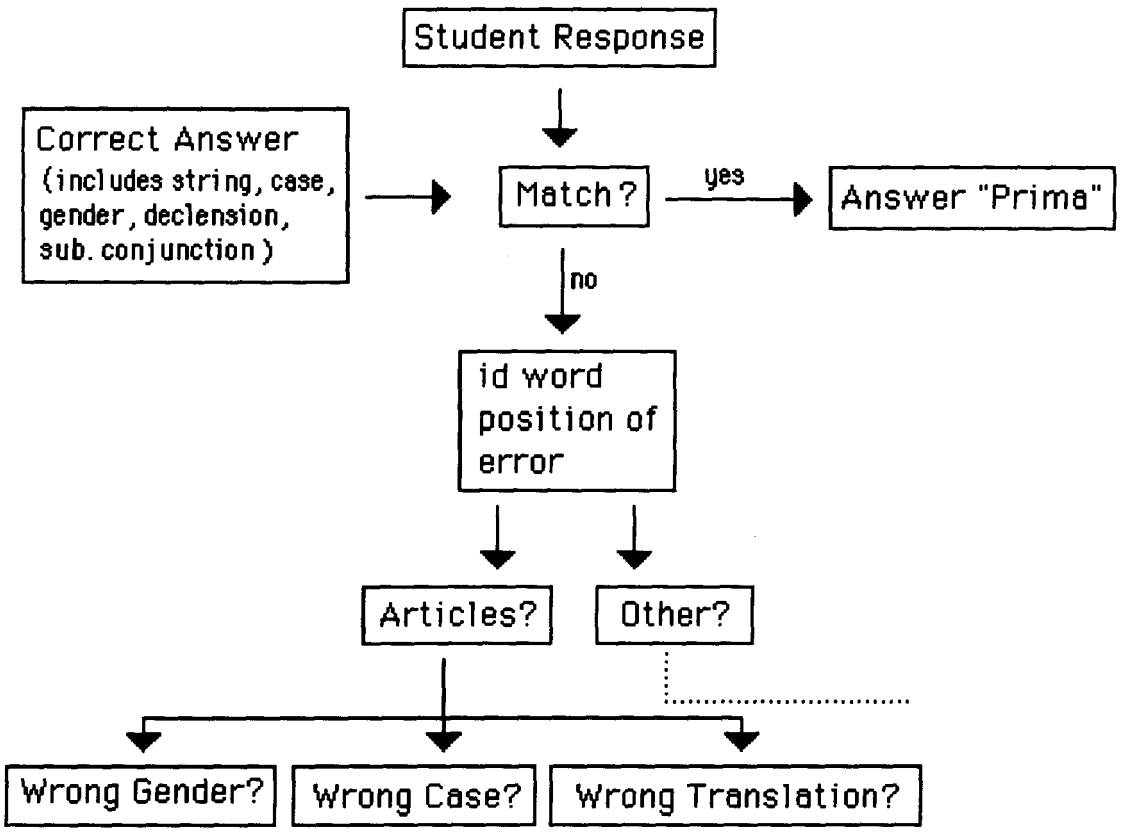

Figure 4-1

After the student has provided the answer s/he clicks the 'DONE' button (Figure 4-4, section 4.4.1) in which the correct answer is specified; that is, the button contains the complete sentence as well as the position of the articles in the string, and a module for the

correct article. This module consists of three parameters: det, artfield, and artcard. While the string match allows the system to quickly scan the student's answer, deeper error analysis occurs if the two sentences do not match. This requires a detailed short-hand description of the articles. In the example given, the following information is stored:

Das Auto gehört **der** Frau.

1. position in string: **1**
   parameter **det**: **3** ('das' is the third article in the field 'detnom')
   parameter **artfield**: **detnom** (name of the field where articles of nominative Case are stored)
   parameter **artcard**: **defarticles** (name of the card where all definite articles are stored)
2. position in string: **4**
   parameter **det**: **2** ('der' is the second article in the field 'detnom')
   parameter **artfield**: **detdat** (name of the field where articles of dative Case are stored)
   parameter **artcard**: **defarticles** (name of the card where all definite articles are stored)

After the program has identified by string match that the student's answer, position 1 in the string, a definite article, is incorrect it enters the first subroutine on articles, the *gender* check (Code example 4-1, line 2-9). Line 2 calls the function *IncorrectInflection* (Code example 4-2) which looks up the student's answer (nominative, masculine) in the field 'detnom'. The system recognizes that the answer provided by the student is in the field

'detnom' although it is not the correct gender. Accordingly, the subroutine responds with: "Wrong gender?" (Code example 4-1, line 5). The incorrect article is also highlighted (Code example 4-1, line 4) to indicate to which word the error message applies.

In the second noun phrase, the student's answer (nominative/ accusative, feminine) is again compared against the correct answer (dative, feminine). In this instance, however, the system does not find the answer provided by the student in the field 'detdat', so the subroutine on *gender* (Code example 4-1, line 2-9) does not apply and the article check continues with the subroutine on *Case* (Code example 4-1, line 10 -19). Line 11 calls the function *IncorrectInflection* (Code example 4-2) to check whether the student's answer is in the field 'cases'. If it is, as in the example, the system responds with: 'Wrong case?'. If not, the system bypasses the subroutine on *Case*, and answers with 'Wrong translation?' (Code example 4-1, line 19). This catches instances where the student provides an indefinite article instead of a definite one.

There are of course instances in German where it is not possible, for either the language instructor or a computer program, to tell whether the student applied the incorrect Case or the incorrect gender. Consider the following example:

Er schreibt mit *der Kuli[8].

He is writing with the pen.

8. 'Kuli' is a masculine noun and therefore the correct sentence is:
Er schreibt mit dem Kuli.

If the student knows that 'Kuli' is masculine, s/he did not apply the dative Case required after the preposition 'mit'. But if the student is under the assumption that 'Kuli' is feminine, s/he applied the right Case.

In such ambiguous instances, the system's default response is: "Wrong case?". This decision, which is reflected in the fixed order of the two subroutines and in the specific organization of the four fields, is based mainly on the author's experience that native speakers of English (which only shows a Case distinction with pronouns) have quite severe problems with the German Case system. The program, therefore, utilizes the idea of native language interference, in the sense that it analyzes errors hierarchically according to their likelihood of occurrence.

The system exhibits a hierarchical order in the error analysis to improve the program's response time as well. The subroutines within the system are ordered in such a way as to search for the most likely errors first. Consider the following student's task:

Verbinden Sie die Sätze. (Join the sentences)

Ich nehme zwei Menüs. Ich habe Hunger. (because)
Ich nehme zwei Menüs, weil ich Hunger habe.

In this example spelling will be checked last since all but one word is given and that word is unlikely to cause spelling problems. Among the most likely errors in this example is use of the correct subordinating conjunction and/or word order.

4.3 Error Message Guidelines

In addition to emphasizing the importance of feedback being informative, Steinberg (1984) also stresses motivation. Feedback and motivation are closely related in the sense that this extrinsic factor provided by an instructor or a computer program propels or prods the student into one or another pattern of behavior. This applies to feedback on errors as well as on correct responses. It even can reach a degree where students may be stimulated to make errors intentionally in order to get the computer's interesting effects on an incorrect answer.

> "The classic anecdote is that in one CAI lesson, the feedback for an incorrect response was a display of a child with tears running down its face. This so fascinated the students that they continually entered incorrect responses to see this display."[9]

At the other end of the spectrum, feedback must not be intimidating: sarcasm, a 'laugh' or an insulting verbal message as a response to an incorrect answer have a discouraging effect on students.

Shneiderman (1987) lists some error message guidelines[10] as an aid to designing informative and motivating feedback. The messages which the system displays to the student follow these guidelines. An example of the system's responses is listed after each:

9. Steinberg, E. R., A Synthesis of Theory, Practice, and Technology, p. 124.
10. Shneiderman, B., Designing the User Interface, p. 320.

1. *Be as specific and precise as possible*
   Wrong gender? Wrong case?
2. *Be constructive: indicate what needs to be done*
   There needs to be a comma separating the main clause from the subordinating conjunction.
3. *Use a positive tone: avoid condemnation*
   Almost! Right verb, right position, but wrong inflection.
4. *Choose user-centered phrasing*
   This button leads you back to your exercise.
5. *Keep consistent grammatical form, terminology, and abbreviations*
   Gender, case, subordinating conjunction, inflection, etc. Abbreviations are not used in the program.
6. *Keep consistent visual format and placement*
   The overall graphics and layout of the program will be discussed in detail in section 4.4.

4.4 Interface

Users undoubtedly prefer one program over another and the judgement is often based on the interactive qualities of a program such as ease of use or functionality.

> "Effective systems generate positive feelings of success, competence, and clarity in the user community. The users are not encumbered by the computer and can predict what happens with each of their actions. When an interactive system is well designed, it almost disappears, enabling the users to concentrate on their work or pleasure. Creating an environment in which tasks are carried out almost effortlessly, requires a great deal of hard work for the designer."[11]

If the designer is aiming at a tool to aid the user in performing tasks, any tool which takes more time and effort to use than the actual task requires is unlikely to be successful; but considering the varying levels of computer literacy, whom are we addressing: novices, intermittent users or experts? Whiteside et al. (1985)[12] compared user interfaces and found that systems that are the easiest to learn are also the easiest to use. While novices may need some introduction to how to use a program and "may need extensive prompting and rely heavily on menus....Experts should be able to shortcut or bypass menus and prompting when desired."[13] The system takes students' various computer backgrounds into account, presenting a menu at the beginning of the program (Figure 4-2).

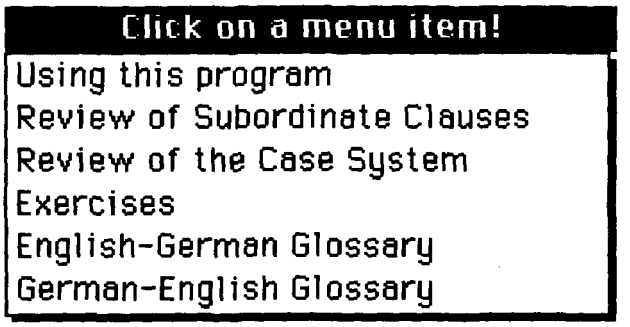

Figure 4-2

When starting the program the student has immediate choices. The novice has the opportunity to become familiar with the program by choosing a two-page introduction; intermittent users, or, experts can

11. Shneiderman, B., Designing the User Interface: Strategies for Effective Human-Computer Interaction, p. 9.
12. Whiteside,J., et al., "User Performance with Command, Menu, and Iconic Interfaces". In Proceedings of CHI '85 Human Factors in Computing Systems, pp. 185-91. New York: Association for Computing Machinery, as cited in Brown, C. M. "Lin", Human-Computer Interface Design Guidelines, p. 13
13. Brown, C. M. "Lin", Human-Computer Interface Design Guidelines, p. 14.

start with the exercises. For students who prefer some theoretical information on the language constructions involved, a review on subordinating conjunctions and the Case system, as well as the relevant vocabulary, can be accessed. Furthermore, within the program an *edit*- and a *help* menu are provided, as illustrated in Figure 4-3.

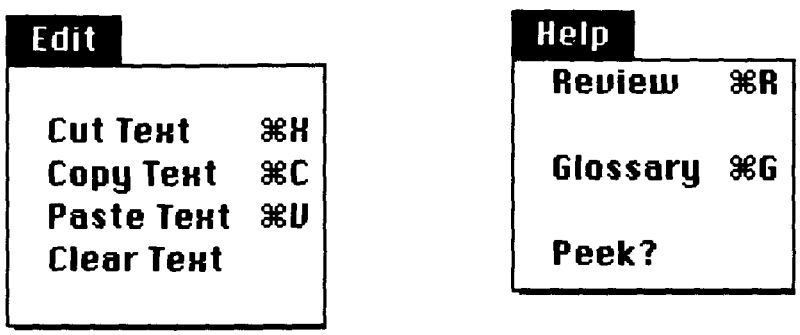

Figure 4-3

The *help* menu supplies the student the opportunity to access additional information in order to find the correct answer[14], while the *edit* menu provides standard Macintosh shortcuts for typing/ correcting the answer in a specified field. While intermittent users and experts may be accustomed to the Macintosh command-key shortcuts, novices might prefer to access the menu items by dragging the menu and clicking the mouse.

4.4.1 Dialog Design

In addition to handling users' diverse levels of computer competence we need to address the overall interface design. Here the

14. "PEEK?" allows the student to look at the correct answer, however, with a deterrent series of prompts.

"*Eight Golden Rules of Dialog Design*"[15] have been extended to suit the specific needs of the language learner.

1. *Strive for consistency*

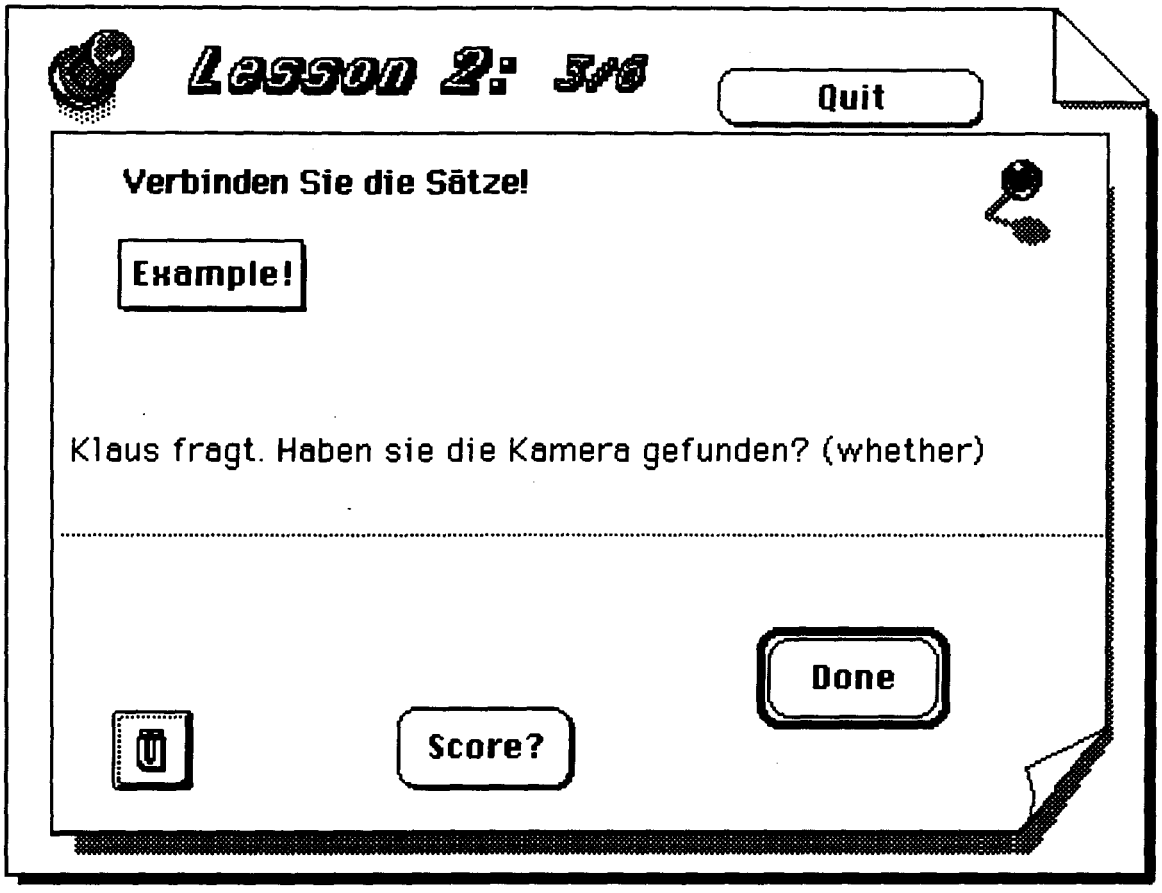

Figure 4-4

Consistency not only applies to feedback, menus, and prompts, but also to pictures/graphics, the overall layout of each page, and the consistent sequences of actions required in similar situations. Figure 4-4 shows the typical functions and graphics of a page in the program. The overall arrangement of each page is identical throughout the

15. Shneiderman, B., Designing the User Interface, p. 61-2.

program: on each page the student finds a marked field to provide his/her answer, and buttons consistent in their function and placement. From a computational point of view, the 'DONE' button does different things on each page but since the intention of the user, to signal completion, is the same throughout the program, the same button is used.

While the principle of consistency with regards to the screen display is especially important to novices, there are also pedagogical considerations relevant to any language student. Upon presentation of an exercise series, an example is displayed in boldfaced type above the exercise. The example is collapsed to a button in subsequent exercises of the same series to minimize the possibility that the student works merely according to a pattern. Access to the example remains, but is now a deterring one step away.

> 2. *Reduce short-term memory load*
> "The limitation of human information processing in short-term memory ("seven plus or minus two chunks") requires that displays be kept simple....Where appropriate, on-line access to command syntax forms, abbreviations, codes, and other information should be provided."[16]

The display intentionally uses a limited number of fields and buttons. Reference to common needed keystrokes is, however, immediately accessible rather than buried in a manual or a help section (Figure 4-5).

In other parts of the program the system handles the minor drudgery of remembering small details. In the review section, for

16. Shneiderman, B., Designing the User Interface, p. 62.

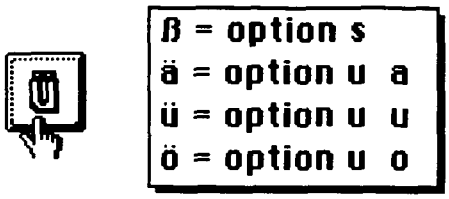

Figure 4-5

instance, once the student has finished reviewing the section on subordinate clauses s/he simply clicks the 'BACK' button to return to the most recent exercise. HyperTalk handles this by the 'Pop Card' command used throughout this system. In HyperCard, information appears on cards which make up a stack. Each card within the stack has an id number. Whenever the student opens a card[17] its id number is stored in a field which is constantly replaced by the id number of the new open card. Use of this device also allows the student to quit the program at any time without having to start the exercises from the beginning again. By entering the name and a password when the student logs in, and by saving this data with the id number of the last card opened, the system is able to identify whether this particular student has logged in before, and allow him/her to continue with the next exercise.

3. *Enable frequent users to use shortcuts*
"As the frequency of use increases, so does the desire to reduce the number of interactions and increase the pace of interaction."[18]

17. This only refers to cards providing the student with an exercise. The id number of other cards, such as for example the Review sections, is not stored.
18. Shneiderman, B., Designing the User Interface, p. 61.

Besides the 'command-key' shortcuts accessing the menu items, the student can also use the <return> or <enter> key to evoke the answer check. While it might be easier for the novice to use the 'DONE' button, frequent users might find it tedious to have to move the mouse repeatedly. An attempt was made to preserve all of the standard Macintosh interface keyboard commands to take advantage of previously acquired user skills, and, to avoid conflicting with user expectations.

4. *Offer simple error handling*

"The user should not have to retype the entire command, but only need repair the faulty part."[19]

This rule is particularly relevant in this program since the student must type a whole sentence, as opposed to a multiple choice scheme where clicking a button suffices. While for pedagogical reasons multiple choice answers are avoided , bad typing skills should not hinder the student in the task. Therefore, if only parts of the student answer are incorrect, the system not only highlights and selects the particular error, positioning the typing cursor (I-beam) at the appropriate location, but also deals with one error at a time. Thus the student need not memorize various errors displayed by several dialog boxes on the screen all at once. As mentioned, all of the standard Macintosh shortcuts (cut, copy, paste) are enabled. Since errors with 'Subordinate Clauses' in German frequently address word order, the student can use these commands to quickly reposition words, without retyping. Consider the example in Figure 4-6:

19. Shneiderman, B., Designing the User Interface, p. 62.

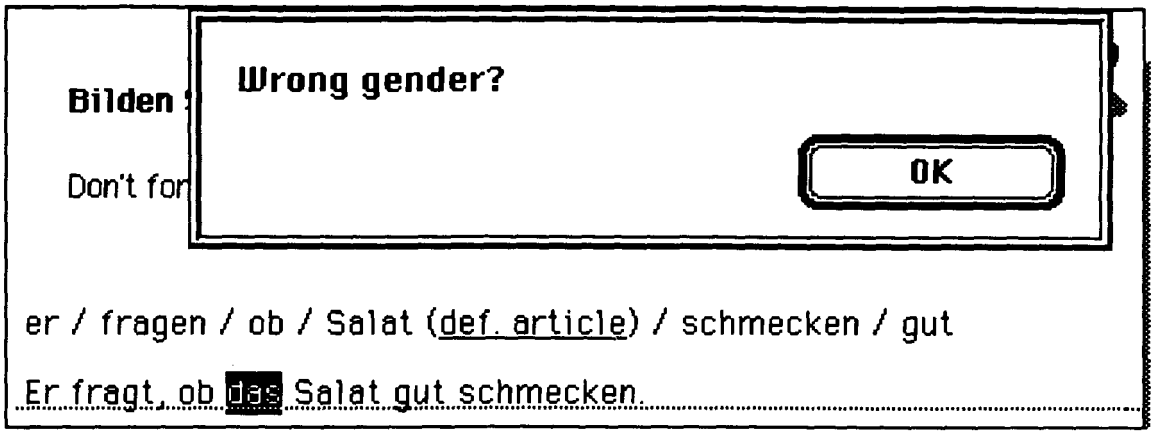

Figure 4-6

The student has provided a wrong article as well as an incorrect subject-verb agreement. The incorrect article is highlighted first. After correcting this error the system will point out the incorrect verb inflection and highlight the verb. Highlighting an error not only points out the specific error but also enables the student to use the <delete> command to quickly remove the word and provide the new answer.

5. *Design Dialogs to yield closure*
"Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operator the satisfaction of accomplishment,..."[20]

While the student is provided with informative feedback after each single exercise, the system also displays the overall score after each lesson, as shown in Figure 4-7:

20. Shneiderman, B., Designing the User Interface, p. 61.

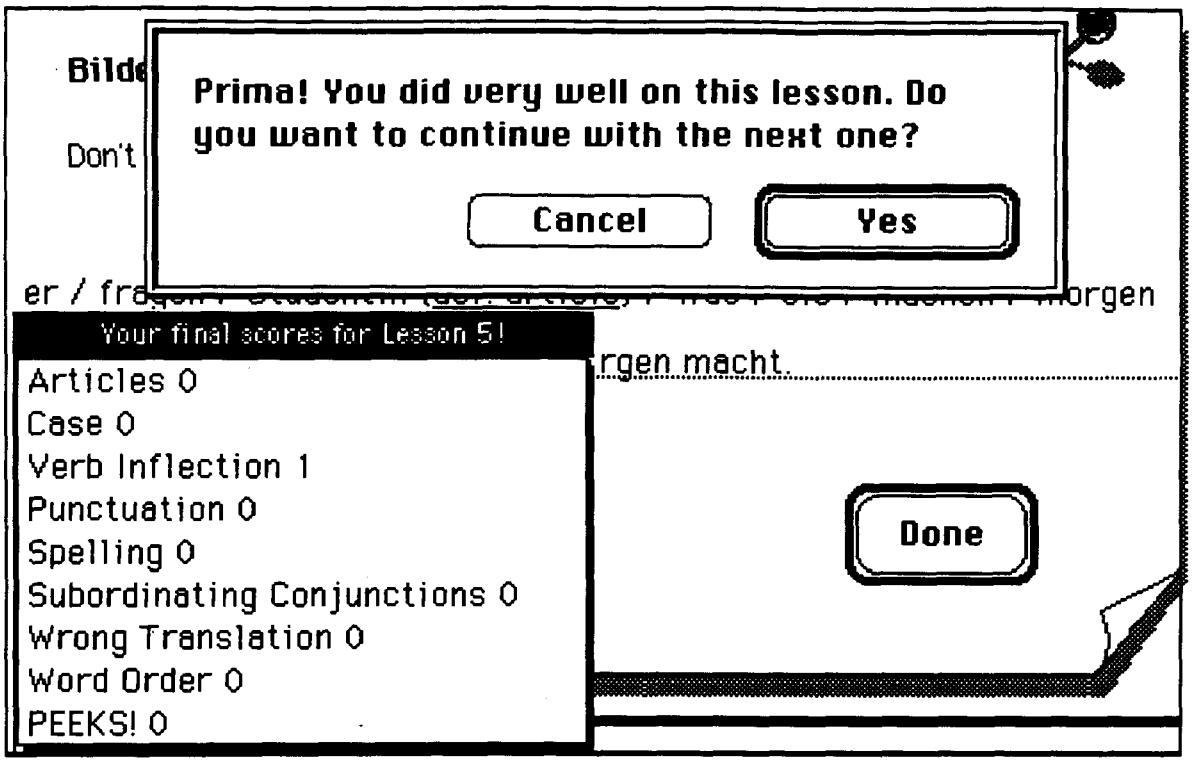

Figure 4-7

The display of the score at the end of a lesson, which consists of various exercises of the same type, gives the student the "satisfaction of accomplishment" and, importantly, informs him/her of his/her progress within the program.

6. *Permit easy reversal of actions*
"This relieves anxiety since the operator knows that errors can be undone, and encourages exploration of unfamiliar options."[21]

This rule mostly applies to novices since the more familiar the student is with a computer the more confident s/he is to explore a program in depth. While in some students anxiety cannot be avoided

21. Shneiderman, B., Designing the User Interface, p. 62.

completely, the program strives to minimize it. First, the introduction to the program tries to familiarize the student with the various functions within the program in a friendly, entertaining way. The student is able to try out the various buttons and menus while explanations of their actions are given. Throughout the program, the 'Quit' button provides the student with a further dialogbox to decide whether s/he really wants to exit the program or not. But even in a case where the student exits the program accidentally the scores and other relevant information are automatically saved. Second, whenever the student leaves, whether intentionally or not, the page of an exercise s/he is working on, a 'BACK' button is provided leading the student back to that page. This applies to all paths open to the student within the program: the Review, Glossary, and score sections. Third, since the student is in control of evoking the answer check, typing errors can be corrected before clicking the 'DONE' button. Effectively, giving the student an infinite number of tries to get to the right answer may reduce anxiety as well. To avoid utter frustration the student can also access the correct answer with the menu item 'peek' under the help menu. The pedagogical rationale behind this decision is to

> "prevent students from guessing, and at the same time to allow them to reconsider their answers and possibly to learn from their mistakes."[22]

However, while the student can access the menu item 'peek' as often as needed, the access is always a deterring three steps away. The system displays three slightly tricky dialogue boxes: the default

22. Steinberg, E. R., Computer-Assisted Instruction, p. 113.

for the first two is 'getting to the answer': in the third one, the default is 'not getting to the answer'. Even if the student escapes the tendency to choose not to peek after all, s/he must switch from the keyboard to the mouse to finally look at the answer. Again, as with the 'example' button described in section 4.4.1 (step 1), by making the access sufficiently inconvenient the student will hopefully make rare use of it. On the other hand the system provides the student with the help s/he might need.

7. *Support internal locus of control*

> "Surprising system actions, tedious sequences of data entries, incapacity or difficulty in obtaining necessary information, and the inability of the action they want all build anxiety and dissatisfaction."[23]

While this rule has already been partly covered above there are further instances where it has been considered in the program[24]. One of the goals of the system is to provide the student with a tool to practice a grammatical construction in German which does not require other material to do the task. The system therefore provides the student with two review sections: one on subordinate clauses and one on the German Case system. The two sections provide the student with the theoretical concepts involved, which are illustrated by examples. In addition, there are two glossaries: English-German, and German-English. Consider Figure 4-8 which illustrates the English-German glossary.

23. Shneiderman, B., Designing the User Interface, p. 62.
24. While an *easy* access to necessary information has been implemented as a general principle of the program, out of pedagogical reasons, there are instances where the access has been made inconvenient, as discussed with the menu item 'peek' and the 'example' button.

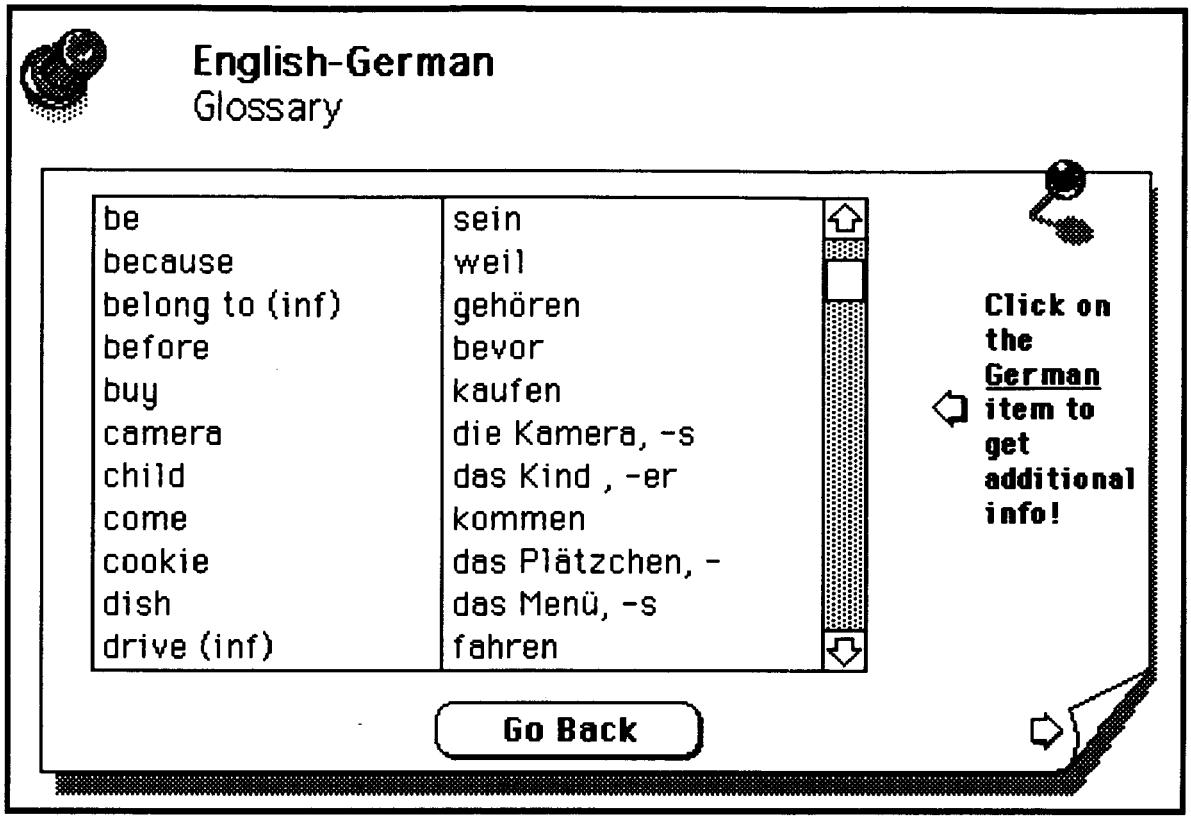

Figure 4-8

The student can access the glossary through the help menu at any time in the program. While the two review sections and the glossaries are essential for working with any language problem, the student also might need grammatical information, such as the conjugation of a verb. It would be merely too tedious and frustrating if a student needed constantly to refer to a book to get the necessary information. Also, if the required help is not easy to access the student will be more likely to guess at the answer, a situation which is undesirable from a language teaching point of view. In this system, the student can access additional information needed through the glossary, as illustrated in Figure 4-9:

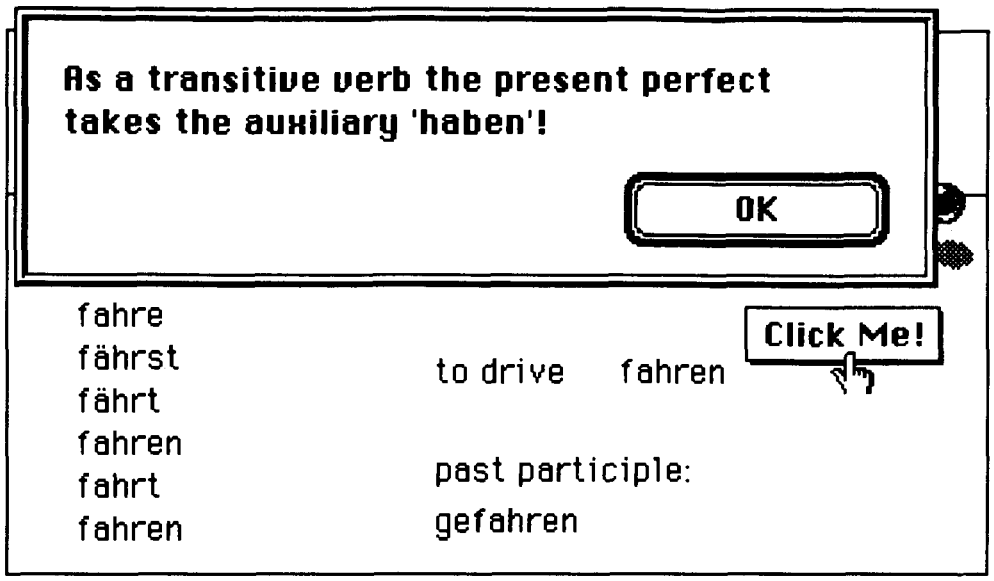

Figure 4-9

After the student clicks the vocabulary item 'fahren' in the glossary s/he receives additional information on this verb: the various inflections, the past participle, and the required auxiliary for the present perfect. In other cases, the student might want to access additional information on nouns, articles, or the Case system. As discussed in section 4.2 the program provides informative feedback to guide the student towards the answer, but does not automatically provide the student with the actual answer. It is therefore important, in order to avoid frustration, that the student be able to access additional information within the program to be able to work independently toward the correct answer.

8. *Offer informative feedback*

"For every operator action there should be some system feedback"[25]

25. Shneiderman, B., Designing the User Interface, p. 61.

To avoid the 'Dead Macintosh' phenomenon, there are instances within the program where system feedback is provided. For example, if the student evokes an answercheck which cannot be supplied at that time[26] the system will respond accordingly. In addition, the system 'beeps' if the cursor is not placed in the answer field and the student tries to type. Additionally, while the system is processing the user's input, checking the answer, the normal MacIntosh system response of setting the cursor to 'watch cursor', or 'busy' cursor is displayed, although, of course, this processing time has been kept to within a matter of a few seconds or less.

26. In Lesson 1, for example, the student is given a paragraph where s/he has to provide the four missing subordinating conjunctions. Even though the instructions are precise it is very possible that a student tries to evoke the answercheck after providing just one of them. In such an instance the system will provide the student with appropriate feedback.

CHAPTER 5

A Theoretical Framework and its Practical Implementation

5.1 Program Design

5.1.1 Optimizing Analysis

The system is designed as an intelligent tutor in contrast to an electronic workbook. The principal advantage of an intelligent tutor is the error-contingent feedback which it provides the student. However, the sheer amount of processing required to provide evaluative feedback can be a disadvantage. The computer platforms on which language instruction programs operate are typically too underpowered to permit the amount of processing required in reasonable time. How can we then use the personal computer efficiently while still providing some semblance of intelligence?

Facts of human communication, noted by H.P. Grice (1967), suggest methods for limiting the depth and/or range of analysis and thus decreasing the processing time. Grice[1] argued that contributions to communication should imply four categories, and adhere to the following maxims:

1. *Quantity*
   'be as informative as required for the current purposes of the exchange'
2. *Quality*
   'try to make your contribution one that is true'
3. *Relation*
   'be relevant'
4. *Manner*
   'be perspicuous'

When applied to responses to students' errors, this analysis suggests, first, that with respect to a particular exercise only errors relevant to that exercise need be analyzed,[2] and second, that detailed linguistic analyses are unnecessary as they contain too much information and would merely burden the student.

5.1.2 The Daemon Approach

The author's implementation of these ideas stresses modularity within the system. The system consists of blocks of exercises which students complete sequentially. Students' work on each exercise is evaluated by a set of daemons; a daemon is a program submodule, typically highly parameterized, which seeks a particular error and takes remedial action when that error is discovered. Each

1. Grice, H.P., "Logic and Conversation". In Cole, P. and Morgan, J. (Eds.) Syntax and Semantics, vol. 3., pp. 41-59
2. For example, in a one word answer, word order is an extraneous concern.

exercise has associated with it a set of daemons which seek errors relevant to the exercise.

To take a specific example, the daemon responsible for ensuring that the subordinating conjunction in a given sentence is correct requires three parameters: a string representing the student's sentence, an integer representing the position of the target subordinating conjunction in the string, and the correct subordinating conjunction. The daemon extracts the word from the string based on the position parameter and compares it with the correct subordinating conjunction. If the words do not match, the daemon takes remedial action.

Consider the following example:

> Verbinden Sie die Sätze. (Join the sentences)
>
> Ich gehe zur Party. Ich habe Zeit. (if)
>
> Ich gehe zur Party, *ob ich Zeit habe.[3]
> I am going to the party if I have time.

The following code checks for the correct subordinating conjunction.

Code example 5-1

```
1  put word spos of string into it
2  if it is not word subconj in card field allsub in card subconjs then
3  if IncorrectInflection(word spos of string, allsub, subconjs) is false then
4  select word RealPosition(punctuation, spos) in field answer1
5  answer "No, '"& it &"' is the wrong subordinating conjunction!"
6  add 1 to sixth word of card field score in card scorecard
7  exit answercheck
8  end if
9  end if
```

3. The correct answer is: Ich gehe zur Party, wenn ich Zeit habe.

The first parameter, 'string', refers to the student answer. 'Spos of string' (line 1) is the specified target position of the subordinating conjunction in the sentence which in the example given is '5'. The correct subordinating conjunction consists of a module which is made up of three parameters: word 'subconj', card field 'allsub', and card 'subconjs'. All subordinating conjunctions are stored in a field of a card in the glossary section which the student can also access by way of reference as outlined in Chapter 4 (section 4.4.1). In our example the parameter *subconj* is specified as '4' ('wenn' is listed as the fourth subordinating conjunction in the field), the parameter *allsub* (the subordinating conjunctions are listed in the field named 'allsub'), and the parameter *subconjs* (the name of the card the subordinating conjunctions are listed in). When checking the student's answer the system compares 'ob' (student's answer) with 'wenn' (correct answer). Since the two do not match, the system responds with: "No, 'ob' is the wrong subordinating conjunction!" (line 5).

The use of parameters within the subroutines is very general in that they can be applied to any word in any position of any sentence; while the subroutine is stored in the common backgrounds of all cards requiring it, the specific information (5, allsub, subconj) is specified in the 'DONE' button of each card. This approach is structurally optimal since we are defining functions to be stored in the system only once, but which can be specified and accessed whenever necessary, using parameters appropriate to the task at hand.

Daemons are, therefore, conceptually simple but capable of simulating intelligence. This combination of simplicity and sophistication is achieved by constraining the domain of the task. In

most cases[4], only one answer is correct and the range of possible errors is small, since the exercises are graded according to difficulty. As discussed in Chapter 3, the program provides the student with bottom-up instruction where the successful completion of a task leads to a slightly more difficult one. This therefore contrasts with domains where the task is highly variable and considerable analysis is required.

5.1.3 Local and Global Daemons

The program employs two kinds of daemons, *local* and *global*. Local daemons may be active or inactive. Active daemons seek errors local to the current exercise and track the frequency of an error in order later to refer students to additional exercises if a threshold is exceeded. Inactive daemons are dormant in the current exercise but can be activated as needed. *Table 5-1: Daemons* gives an overview of the local daemons and shows at which point in the program they are activated.

Local Daemons	active
Articles	Lesson 5, 6
Case	Lesson 5, 6
Verb Inflection	Lesson 2, 5, 6
Punctuation	Lesson 2, 3, 4, 5, 6
Subordinating Conjunctions	Lesson 1, 2, 5, 6
Wrong Translation	Lesson 6
Word Order	Lesson 2, 3, 4, 5, 6

Table 5-1: Daemons

4. In the final game there are more than one possible answer.

While local daemons are directed to specific positions within the sentence, as with the *subordinating conjunction* daemon discussed in section 5.1.2, global daemons seek errors common to all exercises. Typically, these search the entire sentence for errors. An example of a global daemon implemented in the program is the *Spelling* daemon which seeks spelling errors by scanning each word in a sentence.

The overall modularity of the system allows one to create a "pool" for each exercise, adding daemons from previous exercises to the current one. The daemon approach ensures relevance of response: the order in which daemons are activated keeps the point of the current exercise salient. Additionally, the overall system can be easily extended to encompass new phenomena by adding new daemons to the pool.

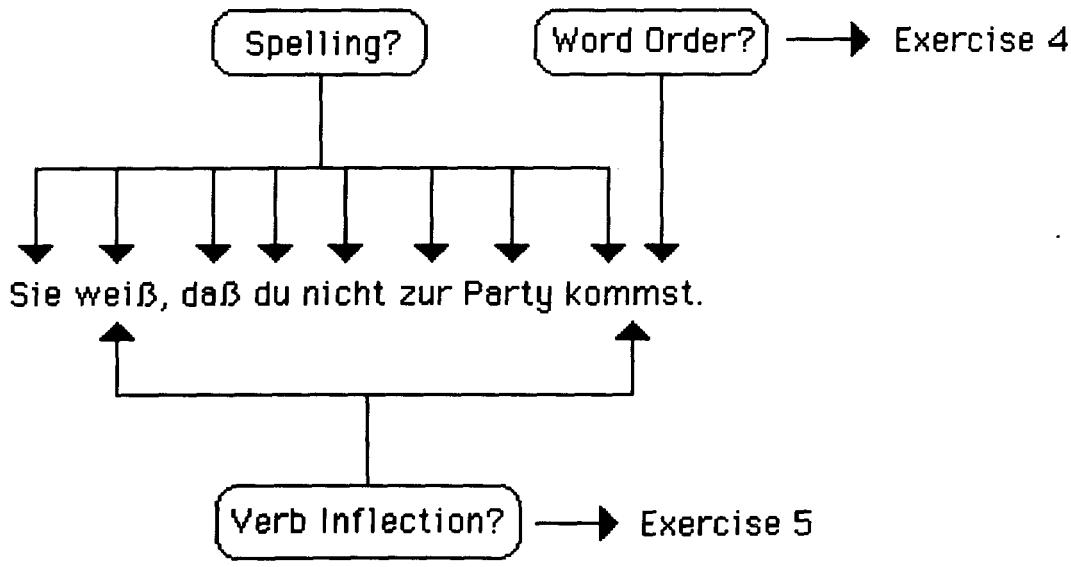

Figure 5-1

Figure 5-1 illustrates the concept of a pool of daemons. Grading the exercise requires checking that the verb of the subordinate clause is in final position, that both verbs are correctly conjugated and that each word is correctly spelled. Moreover, the daemon *Word Order?* redirects the students to Exercise 4 if it discovers that the verb is not sentence final while the daemon *Verb Inflection?* redirects students to Exercise 5 if either of the verbs are incorrectly conjugated. These daemons are local and active. They may also be used in subsequent exercises. The daemon *Spelling?* is global - each exercise inherits it by default.

5.1.4 Error Thresholds

One other advantage of the Daemon approach is that we can provide information about a student's performance by cataloguing the types of errors made. The catalogue is created and maintained by the daemons, each counting the student's errors which it handles. For example, if the *Word Order* daemon responds to an incorrect student answer an error is recorded on the scoresheet[5] as shown in Figure 5-2.

At the end of each block of exercises, the student is presented with a scoresheet reflecting the overall performance within the block. In addition, the values in the scoresheet signal whether or not, and what kind of remedial work is required. Whether remedial work is required for a particular type of error is controlled by establishing a threshold for each error in each block of exercises. If the threshold is exceeded, the student is directed to further work. Figure 5-3 illustrates what a student might encounter at the end of Lessons 5.

5. p. 57, Code example 5-1 (line 5).

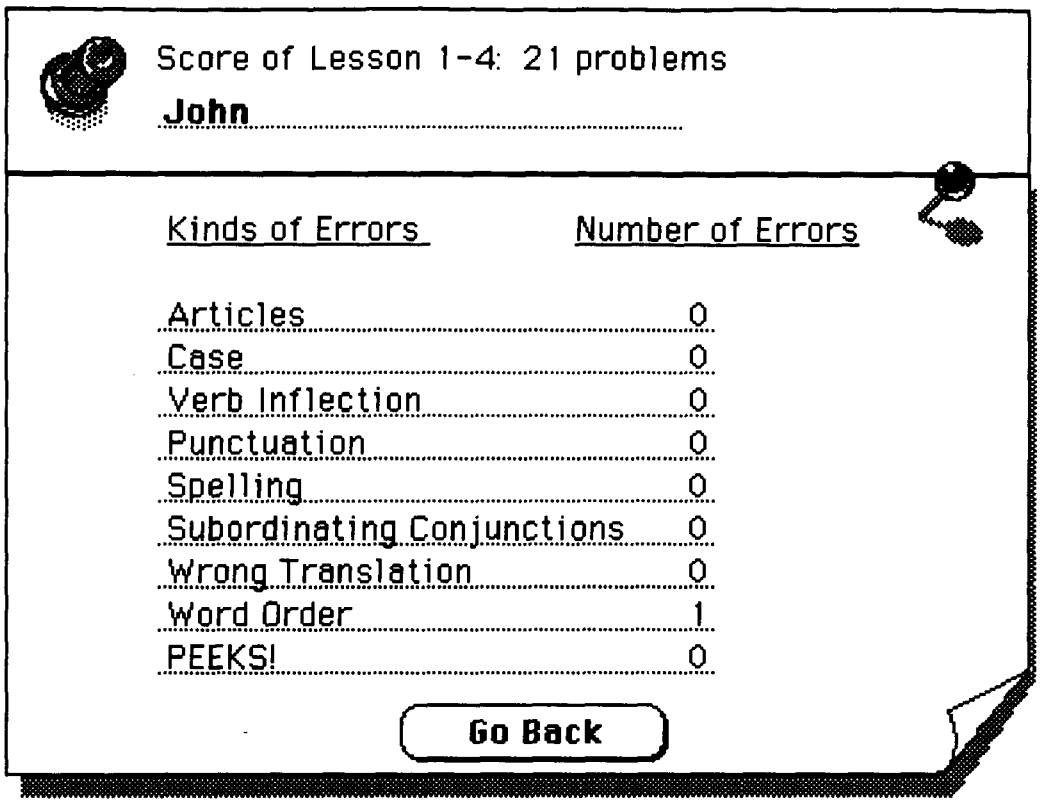

Figure 5-2

Figure 5-3 shows that the student made 4 errors in word order and 4 in spelling. In lesson 5, however, the student's task was to conjugate verbs, supply articles, and provide the correct word order. Since the German words needed to construct each sentence were provided, it is presumed that recorded spelling errors are likely to have represented typos. The threshold for spelling errors is therefore higher than for word order and the student is accordingly only directed to remedial exercises on word order.

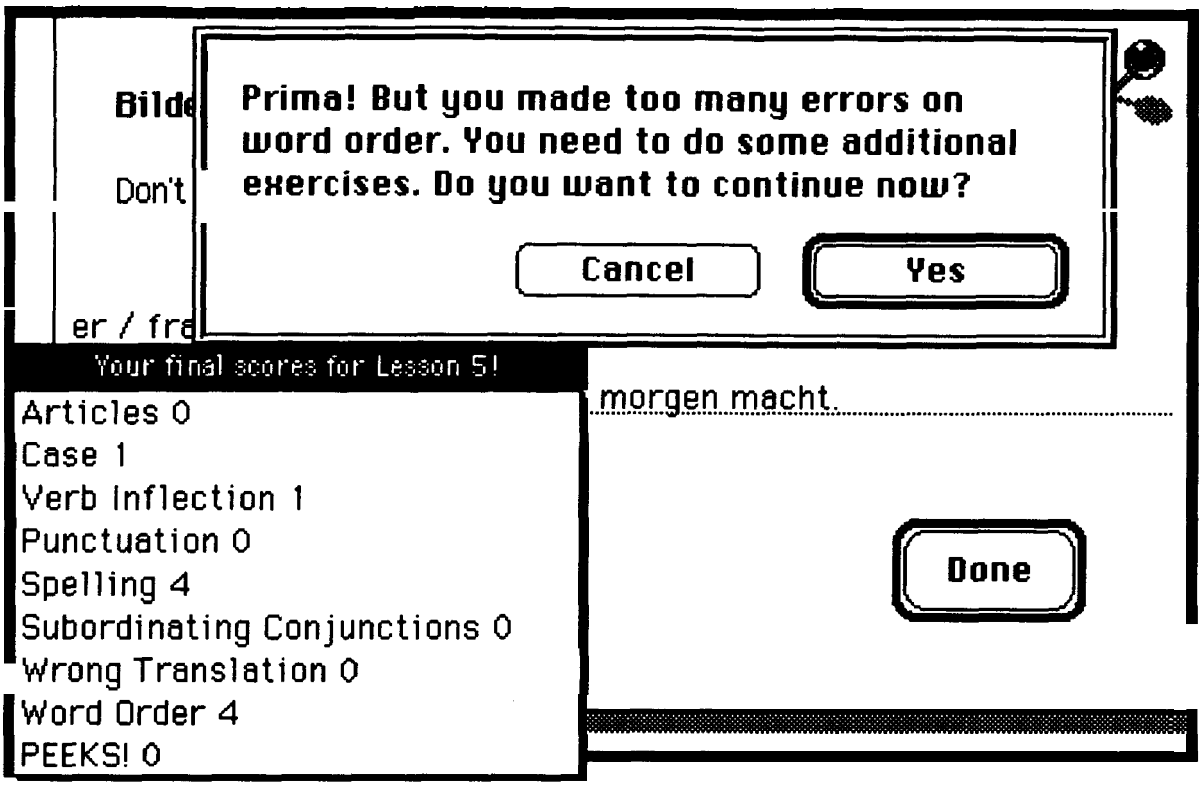

Figure 5-3

5.2 Program Implementation

5.2.1 Daemons in Action

In addition to tracking performance for remedial work, daemons also provide the intelligent, immediate feedback, discussed in Chapter 4, within each exercise. Figure 5-4 shows the response of a local daemon which signals errors in the position of the verb.

In this exercise the student's task is to form a sentence with the words provided. The student's answer activates the *word order*

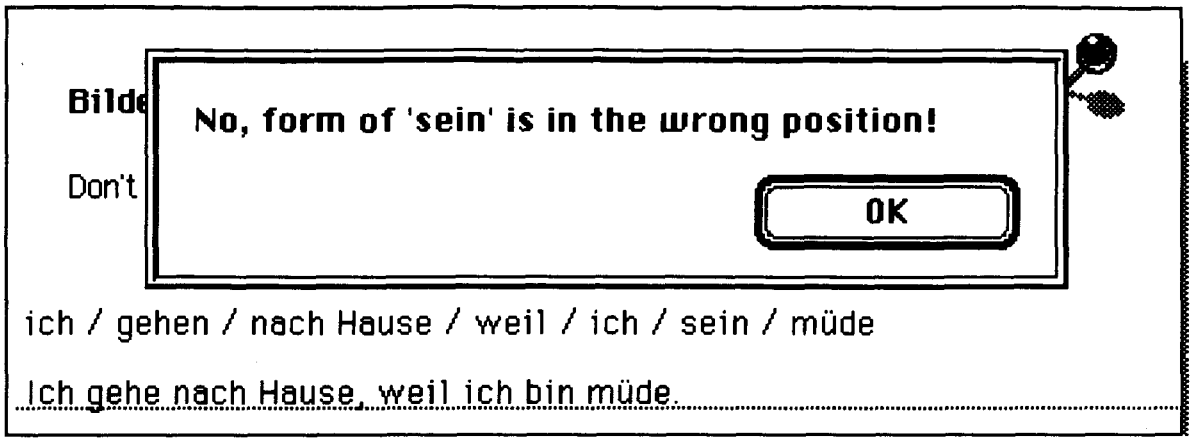

Figure 5-4

daemon because the verb is not in sentence-final position. In Code example 5-2,

Code example 5-2

```
1  repeat with wordposition = 1 to number of words in string
2  if isincard(word wordposition in string, tense, verbcard) is true and ¬
3  wordposition is not vpos then
4  answer "No, form of '"& verbcard & "' is in the wrong position!"
5  add 1 to eighth word of card field score in card scorecard
6  exit answercheck
7  end if
8  end repeat
```

the subroutine (line 1-8) checks with the function *isincard* (line 2) to discover whether the student's answer contains a *possible* inflection of sein. If the student used a possible inflection of 'sein' but the verb is not in the specified position (vpos=8) the system will respond (line 4) as in Figure 5-4. If no possible inflection of 'sein' can be found within the student answer the system will respond with: "This is not a possible inflection of 'sein'! Spelling?" as shown with Code example 5-3 (line 8).

In an alternate answer to the same exercise (Figure 5-5) the student has correctly placed the verb in sentence final position (vpos=8) but provided incorrect subject-verb agreement. In this case the *word order* daemon, although active, remains quiescent but the *verb inflection* daemon responds.

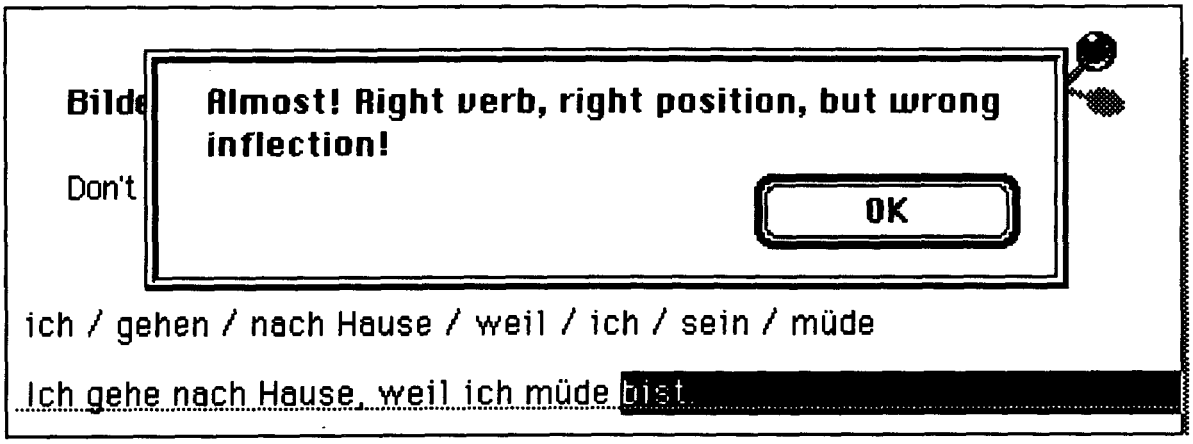

Figure 5-5

Consider the following code:

Code example 5-3

```
1   if word vpos of string is not word verb of card field tense¬
2   in card verbcard then
3   if IncorrectInflection(word vpos of string, tense, verbcard) is false then
4   select word RealPosition(punctposition, vpos) in field answer1
5   answer "Almost! Right verb, right position, but wrong inflection!"
6   add 1 to third word of card field score in card scorecard
7   else
8   answer "This is not a possible inflection of '" & verbcard &"'! Spelling?"
9   add 1 to third word of card field score in card scorecard
10  end if
11  exit answercheck
12  end if
```

This subroutine (line 1-12) applies if the verb the student provided in position 8 is not the correct inflection ('bin') of the verb. The function *incorrectinflection* (line 3)checks whether the student answer is one of the possible inflections of the verb provided in a card field 'tense' in the card 'verb'. As in this case, the system will then respond (line 5) as shown in Figure 5-5. Otherwise the response will be "This is not a possible inflection of 'sein'! Spelling?" (line 8).

The system also has a built-in dictionary which enables it to distinguish a spelling error from an incorrect translation. In the exercise shown in Figure 5-6 the student needed to translate the sentence provided. While all local daemons are satisfied by the answer given, the global daemon *spelling* responds that the student misspelled the second person singular of the verb 'wissen' (weißt).

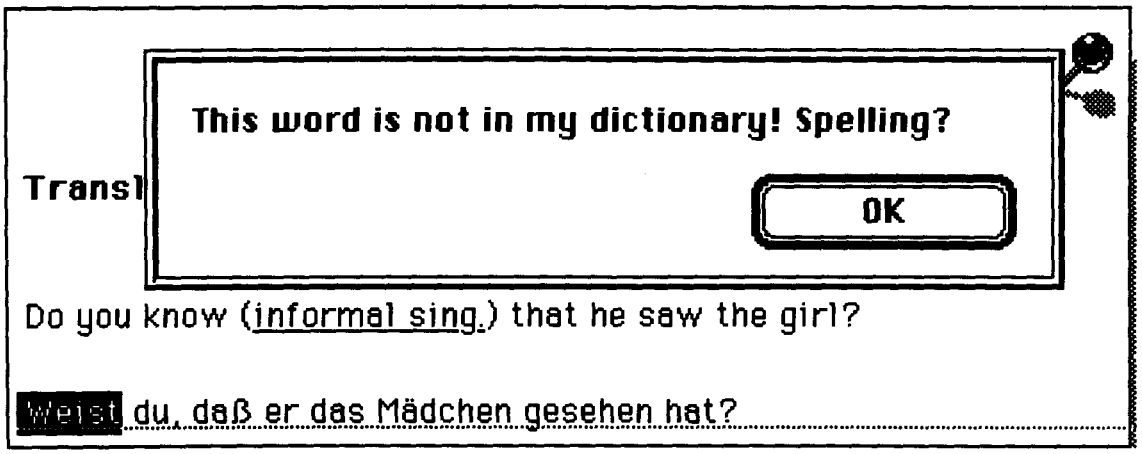

Figure 5-6

Code example 5-4 shows the *indict* function:

Code example 5-4

```
1   if indict(word ok of string, increment) is false then
2   select word RealPosition(punctposition,ok) in field answer1
```

```
3   answer "This word is not in my dictionary! Spelling?"
4   add 1 to fifth word of card field score in card scorecard
5   exit answercheck
6   end if
```

The function *indict* (line 1-6) uses a binary lookup - in the interest of speed - to scan for the word within the built-in glossary. The student answer which through a string match has been identified as incorrect (word ok in string, line 1) cannot be found in the dictionary (Figure 5-6); therefore the student must have misspelled the word s/he used, and the system responds accordingly (line 3).

The processing for the detection of spelling errors in German required a low-level rewrite of the string functions available in Hypertalk since the scripting language, by default, does not distinguish between upper- and lower case nor between Umlauts and their counterpart letters (ü/u, ä/a, ö/o).

The various functions are:

Code example 5-5

```
1   Function Wordcheck Correctanswer, string
2   set cursor to busy
3   put 0 into CorrectIndex
4   put 0 into WordCount
5   repeat with increment = 1 to number of words in string
6   put 1 + WordCount into WordCount
7   put 1 + CorrectIndex into CorrectIndex
8   if increment  1 then
9   if not CorrectCapitalization(word increment in string,word CorrectIndex in CorrectAnswer)
10  then
11  return increment
12  end if
13  end if
14  put word increment of string into it
15  if it is not word CorrectIndex in correctAnswer then
```

```
16  return increment
17  else
18  if CompleteSpellCheck(it, word CorrectIndex in correctAnswer) then
19  next repeat
20  else
21  return increment
22  end if
23  end if
24  end repeat
25  return true
26  end wordcheck

27  Function CorrectCapitalization Word, Correct
28  set cursor to busy
29  If CharToNum(first character of Word) = CharToNum(first character of Correct) then
30  return true
31  else
32  return false
33  end if
34  end CorrectCapitalization

35  Function CompleteSpellCheck Word, Correct
36  set cursor to busy
37  if number of characters in Word = number of characters in Correct then
38  repeat with index = 1 to number of characters in Word
39  put CharToNum(character index of Word) into it
40  if it = 128 or it = 133 or it = 134 or it = 138 or it = 154 or 40 it = 159 or it = 65 or it = 79
41  or it = 85 or it = 97 or it = 111 or it = 117 then
42  if it = CharToNum(character index of Correct) then
43  next repeat
44  else
45  return false
46  end if
47  end if
48  end repeat
49  end if
50  return true
51  end CompleteSpellCheck
```

Each of the three functions- *wordcheck* (line1-26), *correct-capitalizationword* (line 27-34), and *completespellcheck* (line 35-51) - contains a loop and two parameters. In *wordcheck,* the system first

compares words: the student's answer (*string*) with the correct word (*correctanswer*). Since Hypertalk is incapable of recognizing the difference between, for example, 'gute' and 'Güte', the two functions *correctcapitalizationword* (line 9) and *completespellcheck* (line 10) are necessary. These functions both have two parameters, *word* and *correct,* where *word* refers to the student answer and *correct* to the correct answer. While *correctcapitalization* looks at the first letter of the words for lower/upper case *completespellcheck* checks each letter of a word (Umlauts are specified by their ASCII characters). If the student answer deviates from the correct answer the student receives feedback that a spelling error occurred.

In contrast to spelling errors, Figure 5-7 shows a case where the student has provided a verb which exists in the dictionary. It is spelled and inflected correctly but is, nonetheless, an incorrect translation of 'to know'.[6] After the spellchecker finds the student answer in the dictionary the system runs through the *IncorrectInflection/Wrong Translation* subroutines. Code example 5-6 shows the actual functions at work:

Code example 5-6

```
1  if word vpos of string is not word verb of card field tense in card verbcard then
2  if IncorrectInflection(word vpos of string, tense, verbcard) is false then
3  select word RealPosition(punctposition, vpos) in field answer1
4  answer "Almost! Right verb, right position, but wrong inflection!"
5  add 1 to third word of card field score in card scorecard
6  else
7  select word RealPosition(punctposition, vpos) in field answer1
8  answer "This is not the correct translation! Spelling?"
9  add 1 to third word of card field score in card scorecard
```

6. This is a very common mistake made by English native speakers since German, as do many other languages, has two distinct verbs for the verb 'to know'.

```
10  end if
11  exit answercheck
12  end if
```

The first part of the subroutine (line 2-5), *Incorrectinflection*, does not respond because the student answer is none of the possible inflections of 'wissen'. And since the student provided a word which is in the dictionary the *Spelling* daemon and the *Verb Inflection* daemon pass it on to the daemon *Wrong Translation* as shown in Figure 5-7:

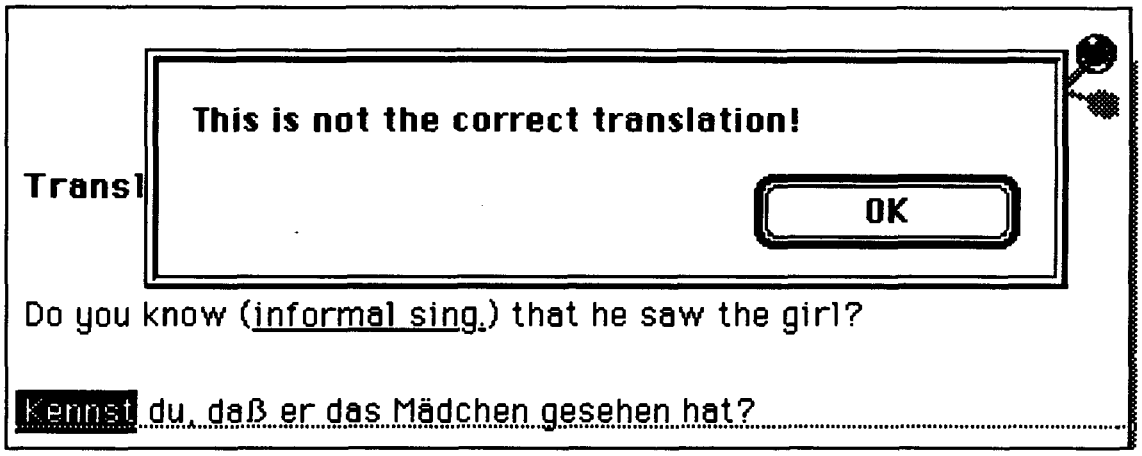

Figure 5-7

From a language teaching point of view the distinction between these two errors is necessary and important enough to build in to the system, even at the cost of a slower response time. In the cases illustrated by Figure 5-6 and Figure 5-7, not receiving error-contingent feedback the student would not only be confused but would also remain unaware of the source of error. In Figure 5-6 the user needs to practice the inflections of 'wissen' while in Figure 5-7 the important distinction between 'wissen'/'kennen' has to be part of a

follow-up activity. These two remedial exercises are not interchangeable if our goal of language teaching is accurate speech.

CHAPTER 6 Conclusion

As the software accompanying this project demonstrates, Intelligent Tutoring Systems are fully realizable on micro computer platforms. Although there is no reason to assume that the program could not have been successfully implemented on some other system, the choice of the Macintosh computer with its user-friendly interface seems to have been optimal. In addition, given the author's limited previous computer programming experience, Apple's HyperCard more than justified its claim to be a powerful, yet relatively easy to use authoring system. For all its points, however, HyperCard will not be the author's preferred environment for future work; past a certain level of complexity a more flexible and powerful programming language is required.

In the rapidly changing world of computers one can be optimistic that hardware will continue to become cheaper, faster, and more powerful. In other words, the technology will continue to

improve, allowing the software to become even more effective and elaborate. Many new developments, such as CD Interactive and to an extent CD-ROM, are of particular interest to CALL designers inasmuch as the incorporation of sound and image could greatly enhance existing software.

Most of the above conclusions were drawn following a very encouraging beta test conducted in April 1993 at Simon Fraser University. 24 participants, the combination of students from 2 introductory German classes, went through the entire program, the quickest students requiring about 50 minutes, and only 1 student failing to complete all exercises within one and a half hours. Their evaluations of the program[1] were positive and some minor bugs (the inevitable small flaws) were identified and subsequently corrected. Since the participants were volunteers and had already been imposed upon, to some extent, by the beta test itself, there was no rigorous after-testing to assess the program's efficacy in teaching subordinate clauses. The responses to the questionnaires, however, indicate that the students, at least subjectively, felt the program to be effective. Interestingly, while the testees could be expected to praise the software (in the immediate presence of the author), the exact nature of their comments paralleled the theoretical rationale underlying its design. That is, they appreciated precisely those aspects of the program, the error-contingent feedback, the individual tailoring, etc. which were earlier stressed as being of fundamental importance to CALL. Specifically, students found it more motivating than a workbook, enjoyed working at their own pace, and were

1. see Appendix B for a sample questionnaire.

gratified by its ease of use, appropriate and instant feedback, and the on-line access to review. One student, for instance, wrote,

- We all have individual mistakes and this program gives me instant feedback as to what's wrong with my sentences.

Another responded,

- Excellent feedback! The best was: "Almost..." or "I won't count that." It was very useful.

One point raised by the students' responses of some interest to future development, and not previously discussed here, is the desirability of incorporating sound within the program. In actual fact, this was implemented in a small way in the review and introduction sections of the program. By clicking a sound button students could listen to a recording of the author speaking the subordinating conjunctions and so on. Many of the respondents indicated that they would have liked sound to have been incorporated throughout the exercises. The sole reason this was not done was one of purely practical disk-storage limitations. A more extensive audio component would have quickly required more storage space than is available on a floppy disk. The obvious advantages of incorporating sound are exposure to pronunciation, stress and intonation. Sound would most certainly be desirable in a full-blown version of the current software and is realizable with present technology.

Program Listing

```
1.  on openField            -- disable arrow keys
2.  global TextEntry
3.  put true into TextEntry
4.  send "openField" to Hypercard
5.  end openField

6.  on mouseDown
7.  global TextEntry
8.  put false into TextEntry
9.  end mouseDown

10. on arrowkey whichkey
11. global TextEntry
12. if textEntry is true then
13. if whichkey = "left" then
14. send "arrowkey left" to Hypercard
15. end if
16. if whichkey = "right" then
17. send "arrowkey right" to Hypercard
18. end if
19. end if
20. end arrowkey

21. on openstack            -- set up menu and menuitems
22. delete menu "Style"
23. delete menu "Font"
24. delete menu "Go"
25. delete menuitem "undo" of menu "edit"
26. delete menuitem "new card" of menu "edit"
27. delete menuitem "delete card" of menu "edit"
28. delete menu "File"
29. create menu "Help"
30. put "Review" into menu "Help"
31. set the cmdchar of menuitem "Review" of menu "help" to R
32. put "--" after menuitem "Review" of menu "Help"
33. put "Glossary" after the second menuitem of menu "Help"
```

```
34. set the cmdchar of menuitem "Glossary" of menu "help" to G
35. put "--" after menuitem "Glossary" of menu "Help"
36. put "Peek?" after the fourth menuitem of menu "Help"
37. set textarrows to true
38. ask "Please enter your name"          -- login
39. if it is empty then
40. ask "You need to enter your name"
41. if it is empty then
42. domenu "quit Hypercard"
43. exit openstack
44. end if
45. end if
46. put it into card field name in card scorecard
47. ask password "Please enter your password"
48. set cursor to busy
49. put it into card field password in card scorecard
50. repeat with increment = number of lines in card field scorehistory in card scorecard
    down to 1
51. if line 1 in card field name in card scorecard is first word in line increment in card
    field scorehistory in card scorecard and line 1 in card field password in card
    scorecard is third word in line increment in card field scorehistory in card
    scorecard and word 5 in line increment in card field scorehistory in card scorecard =
    "completed" then
52. answer "You've done the exercises before, which one would you like to review?" with
    "Lesson 6" or "Lesson 5" or "Lesson 1 - 4"
53. set cursor to busy
54. if it is "Lesson 1 - 4" then
55. delete line increment in card field scorehistory in card scorecard
56. visual effect iris open
57. go to card id 16497
58. tabkey
59. put "0 0 0 0 0 0 0 0 0" into card field score in card scorecard
60. end if
61. if it is "Lesson 5" then
62. delete line increment in card field scorehistory in card scorecard
63. visual effect iris open
64. go to card id 20286
65. tabkey
66. put "0 0 0 0 0 0 0 0 0" into card field score in card scorecard
67. end if
68. if it is "Lesson 6" then
69. delete line increment in card field scorehistory in card scorecard
70. visual effect iris open
```

```
71.  go to card id 12850
72.  tabkey
73.  put "0 0 0 0 0 0 0 0 0" into card field score in card scorecard
74.  end if
75.  exit openstack
76.  else
77.  if line 1 in card field name in card scorecard is first word in line increment in card
     field scorehistory in card scorecard and line 1 in card field password in card
     scorecard is third word in line increment in card field scorehistory in card
     scorecard then
78.  answer "Do you want to continue where you stopped last time?" with "no" or "yes"
79.  set cursor to busy
80.  put word 15 to 16 of line increment in card field scorehistory in card scorecard into
     saved
81.  if it is "yes" then
82.  set cursor to busy
83.  visual effect iris open
84.  go card saved
85.  tabkey
86.  put word 5 to 13 of line increment in card field scorehistory in card scorecard into
     line 1 of card field score of card scorecard
87.  delete line increment in card field scorehistory in card scorecard
88.  end if
89.  if it is "no" then
90.  put "0 0 0 0 0 0 0 0 0" into card field score in card scorecard
91.  delete line increment in card field scorehistory in card scorecard
92.  end if
93.  exit openstack
94.  else
95.  end if
96.  end if
97.  end if
98.  end repeat
99.  end openstack

100. on closestack
101. set cursor to busy
102. delete word 1 in card field id 4 in card id 16497        -- clear fields in Lesson 1
103. delete word 1 in card field id 5 in card id 16497
104. delete word 1 in card field id 6 in card id 16497
105. delete word 1 in card field id 7 in card id 16497
106. repeat with increment = 18 to the short number of card 66  --clear all other fields
```

```
107. if line 1 in field answer1 in card increment is empty then
108. next repeat
109. else
110. if line 1 in field answer1 in card increment is not empty then
111. delete line 1 in field answer1 in card increment
112. if the short number of this card = 66 then
113. exit repeat
114. end if
115. end if
116. if line 1 in field answer1 in card increment is empty and line 1 in field answer1 in
     card increment + 1 is empty then
117. exit repeat
118. end if
119. end if
120. end repeat
121. if short id of this card = 32254 then        -- clear lesson 1 if student did not do any
                                                    -- further exercise
122. delete line 1 in card field name in card scorecard
123. delete line 1 in card field password in card scorecard
124. delete line 1 in card field score in card scorecard
125. delete line 1 in card field ids in card scorecard
126. exit closestack
127. else
128. if short id of this card = 43357 then      -- student completed the whole program
129. put (number of lines in card field scorehistory in card scorecard) + 1 into it
130. put (card field name in card scorecard) && "/" && (card field password in card
     scorecard) && "/" && (card field ids in card scorecard) && "--" & the short date
     into line it in card field scorehistory in card scorecard
131. exit closestack
132. else
133. put (number of lines in card field scorehistory in card scorecard) + 1 into it
134. put (card field name in card scorecard) && "/" && (card field password in card
     scorecard) && "/" && (card field score in card scorecard) && "/" && (line 1 in
     card field ids in card scorecard) && "--" & the short date into line it in card field
     scorehistory in card scorecard
135. end if
136. end if
137. end closestack

138. Function showscore
139. set cursor to busy                          -- popup field with score
140. show card button 2 in this card
141. put word 1 of card field score in card scorecard into it
```

142. put (word 1 of line 1 of card field 2 in this card) && (it) into line 1 in card field 2 in this card
143. put word 2 of card field score in card scorecard into it
144. put (word 1 of line 2 of card field 2 in this card) && (it) into line 2 in card field 2 in this card
145. put word 3 of card field score in card scorecard into it
146. put (word 1 to 2 of line 3 of card field 2 in this card) && (it) into line 3 in card field 2 in this card
147. put word 4 of card field score in card scorecard into it
148. put (word 1 of line 4 of card field 2 in this card) && (it) into line 4 in card field 2 in this card
149. put word 5 of card field score in card scorecard into it
150. put (word 1 of line 5 of card field 2 in this card) && (it) into line 5 in card field 2 in this card
151. put word 6 of card field score in card scorecard into it
152. put (word 1 to 2 of line 6 of card field 2 in this card) && (it) into line 6 in card field 2 in this card
153. put word 7 of card field score in card scorecard into it
154. put (word 1 to 2 of line 7 of card field 2 in this card) && (it) into line 7 in card field 2 in this card
155. put word 8 of card field score in card scorecard into it
156. put (word 1 to 2 of line 8 of card field 2 in this card) && (it) into line 8 in card field 2 in this card
157. put word 9 of card field score in card scorecard into it
158. put (word 1 of line 9 of card field 2 in this card) && (it) into line 9 in card field 2 in this card
159. send "mousedown" to card button 2 in this card
160. end showscore

161. **Function storescore**
162. set cursor to busy -- store score in scorecard
163. put (number of lines in card field scorehistory1 in card scorecard1) + 1 into lines
164. get card field name in card scorecard
165. put it into first word in line lines in card field scorehistory1 in card scorecard1
166. get card field score in card scorecard
167. put (first word in line lines in card field scorehistory1 in card scorecard1) && it && "-" && the short date && "- score" && (first line of card field lesson in card scorecard) & "! !!!!" into line lines in card field scorehistory1 in card scorecard1
168. end storescore

169. **Function NumErrorCheck ErrorIds, myerrormessages, CardIds, Thresholds, msgStatus**
170. set cursor to busy
171. put false into myerrormessage -- set up error messages

```
put false into cardid
put false into errorsFound
repeat with index = 1 to the number of words in ErrorIds
if  word word index of ErrorIds of card field score in card scorecard > word index of Thresholds then
if word index of msgStatus is true then
put true into errorsFound
if myerrormessage = false then
put Makemyerrormessage(myerrormessages, index) into myerrormessage
else
if "and" is in myerrormessage then
put Makemyerrormessage(myerrormessages, index) & "," & myerrormessage into myerrormessage
else
put Makemyerrormessage(myerrormessages, index) & ", and" & myerrormessage into myerrormessage
end if
end if
if cardid is false then
put word index of CardIds into cardid
else
put  word index of CardIds into it
if it is word index - 1 of CardIds then
next repeat
else
put it into card field numberofexercises in card id 35259
end if
end if
else
if errorsFound is false then
put Makemyerrormessage(myerrormessages, index) into myerrormessage
put word index of CardIds into cardid
end if
end if
end if
end repeat
if myerrormessage is not false then
if errorsFound is true then
if short id of this card = 28582 then     -- after lesson 6 if threshold is exceeded
answer "Prima! You made quite a number of errors on" && myerrormessage &". You can move on to the last task, you might want to consider reviewing these exercises at some point. Do you want to continue now?" with "Cancel" or "Yes"
```

```
209. if it is "yes" then
210. do storescore()
211. visual effect dissolve fast
212. go card id cardid
213. put "0 0 0 0 0 0 0 0 0" into card field score in card scorecard
214. end if
215. exit numerrorcheck
216. end if                          -- after any lesson if threshold is exceeded
217. answer "Prima! But you made too many errors on" && myerrormessage &". You need
     to do some additional exercises. Do you want to continue now?" with "Cancel" or "Yes"
218. else
219. answer myerrormessage with "Cancel" or "Yes"
220. end if
221. if it is "yes" then
222. do storescore()
223. visual effect dissolve fast
224. go card id cardid
225. tabkey
226. put "0 0 0 0 0 0 0 0 0" into card field score in card scorecard
227. end if
228. else
229. if short id of this card = 28582 then    -- after lesson 6 if no threshold is exceeded
230. answer "Prima! You did very well on this lesson. Do you want to continue with the
     last one?" with "Cancel" or "Yes"
231. if it is "yes" then
232. do storescore()
233. visual effect dissolve fast
234. go to card id 43357
235. tabkey
236. put "0 0 0 0 0 0 0 0 0" into card field score in card scorecard
237. end if
238. else                            -- after each lesson if no threshold is exceeded
239. answer "Prima! You did very well on this lesson. Do you want to continue with the
     next one?" with "Cancel" or "Yes"
240. if it is "yes" then
241. do storescore()
242. visual effect dissolve fast
243. go to next card
244. tabkey
245. put "0 0 0 0 0 0 0 0 0" into card field score in card scorecard
246. end if
247. end if
```

```
248. end if
249. end NumErrorCheck

250. Function Makemyerrormessage myerrormessages, myid
251. set cursor to busy
252. put 1 into count                          -- error messages for peeks
253. put "" into myerrormessage
254. repeat with index = 1 to number of words in myerrormessages
255. put word index of myerrormessages into it
256. if it = peekmsg then
257. put "Prima! But you've taken too many peeks! You need to make up these exercises.
     Do you want to continue now?" into myerrormessage
258. end if
259. return myerrormessage
260. exit Makemyerrormessage
261. end if
262. if myid = count then
263. if it is "|" then
264. return myerrormessage
265. else
266. put myerrormessage && it into myerrormessage
267. end if
268. else
269. if it is "|" then
270. put count + 1 into count
271. end if
272. end if
273. end repeat
274. return myerrormessage
275. end Makemyerrormessage

276. Function Capitalized word
277. set cursor to busy                         -- spellchecker for capitalization
278. put CharToNum(first character of word) into it
279. if it >= 65 and it <= 90 then
280. return true
281. else
282. if it >= 128 and it <= 134 then
283. return true
284. else
285. return false
```

```
286. end if
287. end if
288. end Capitalized

289. Function IsPunctuation character
290. set cursor to busy                    -- check punctuation
291. return character is "," or character is "." or character is "?" or character is "!"
292. end IsPunctuation

293. Function FindPunctuation sentence
294. set cursor to busy                    -- check punctuation
295. repeat with index = 1 to number of words in sentence
296. if IsPunctuation(word index of sentence) then
297. return index
298. end if
299. end repeat
300. return 10000
301. end FindPunctuation

302. Function NormalizeSentence sentence
303. set cursor to busy                    -- error messages for punctuation
304. put first character of sentence into it
305. put false into LastCharacterIsSpace
306. put number of characters in sentence into Length
307. repeat with index = 2 to Length
308. put character index of sentence into NewChar
309. if IsPunctuation(NewChar) then
310. if CharToNum(character index + 1 of sentence) = 32 then
311. if LastCharacterIsSpace then
312. select character index in field answer1
313. answer "Punctuation should not be preceded by a space."
314. return false
315. end if
316. if index = Length then
317. next repeat
318. else
319. put NumToChar(32) into NewChar
320. end if
321. else
322. if CharToNum(character index of sentence) = 44 then
323. select character index in field answer1
```

```
answer "Punctuation should be followed by a space."
return false
else
if CharToNum(character index of sentence)  44 then
next repeat
end if
end if
end if
end if
if ChartoNum(NewChar) = 32 then
if index = Length then
next repeat
else
if LastCharacterIsSpace then
next repeat
else
put it & NewChar into it
put true into LastCharacterIsSpace
end if
end if
else
put it & NewChar into it
put false into LastCharacterIsSpace
end if
end repeat
return it
end NormalizeSentence

Function CompleteSpellCheck Word, Correct
set cursor to busy                    -- spellchecker for 'Umlaut'
if number of characters in Word = number of characters in Correct then
repeat with index = 1 to number of characters in Word
put CharToNum(character index of Word) into it
if it = 128 or it = 133 or it = 134 or it = 138 or it = 154 or it = 159 or it = 65 or
   it = 79 or it = 85 or it = 97 or it = 111 or it = 117 then
if it = CharToNum(character index of Correct) then
next repeat
else
return false
end if
end if
```

```
363. end repeat
364. end if
365. return true
366. end CompleteSpellCheck

367. Function Wordcheck Correctanswer, string
368. set cursor to busy                    -- complete word check
369. put 0 into CorrectIndex
370. put 0 into WordCount
371. repeat with increment = 1 to number of words in string
372. put 1 + WordCount into WordCount
373. put 1 + CorrectIndex into CorrectIndex
374. if increment  1 then
375. if not CorrectCapitalization(word increment in string, word CorrectIndex in
     CorrectAnswer) then
376. return increment
377. end if
378. end if
379. put word increment of string into it
380. if it is not word CorrectIndex in correctAnswer then
381. return increment
382. else
383. if CompleteSpellCheck(it, word CorrectIndex in correctAnswer) then
384. next repeat
385. else
386. return increment
387. end if
388. end if
389. end repeat
390. return true
391. end wordcheck

392. Function CorrectCapitalization Word, Correct
393. set cursor to busy                    -- spellchecker for capitalization
394. If CharToNum(first character of Word) = CharToNum(first character of Correct)
     then
395. return true
396. else
397. return false
398. end if
399. end CorrectCapitalization
```

```
400. Function IncorrectInflection reply, tense, verbcard
401. set cursor to busy                    -- check for inflection
402. put true into it
403. repeat with increment = 1 to number of words in card field tense in card verbcard
404. if reply is word increment in card field tense in card verbcard then
405. return false
406. end if
407. end repeat
408. return it
409. End IncorrectInflection

410. Function isincard answer,tense, verbcard
411. set cursor to busy                    -- check for word position in string
412. repeat with increment = 1 to number of words in card field tense in card verbcard
413. if answer = word increment in card field tense in card verbcard then
414. return true
415. end if
416. end repeat
417. return false
418. End isincard

419. Function indict reply, index, low, high      -- spellchecker
420. set cursor to busy
421. put false into it
422. put 1 into low
423. put (number of words in card field vocab in card vocab) + 1 into high
424. put (high + low) div 2 into new
425. repeat while new is not lastnew
426. put new into lastnew
427. put word(new) in card field vocab of card vocab into her
428. if SameWord(reply, her, index) then
429. put true into it
430. return it
431. exit indict
432. else
433. if reply > her then
434. put new into low
435. put (high + low) div 2 into new
436. else
437. if reply < her then
```

```
438.  put new into high
439.  put (high + low) div 2 into new
440.  end if
441.  end if
442.  end if
443.  end repeat
444.  return it
445.  end indict

446.  Function SameWord reply, correct, index
447.  set cursor to busy   -- spellchecker (ignore capitalization for first word in string)
448.  if reply = correct then
449.  if index = 1 then
450.  return true
451.  else
452.  if reply > correct then
453.  return false
454.  else
455.  if correct > reply then
456.  return false
457.  else
458.  return true
459.  end if
460.  end if
461.  end if
462.  end if
463.  return false
464.  end SameWord

465.  Function RealPosition punctposition, position
466.  set cursor to busy                -- determine position of punctuation in string
467.  if punctposition <= position then
468.  return position + 1
469.  else
470.  return position
471.  end if
472.  end RealPosition
```

Questionnaire

1) Bearing in mind that this program is not intended to substitute for the teacher, but rather is meant as an alternative to workbook review, what do you think are advantages/disadvantages of this computer program?

2) In terms of the program's feedback (response to errors, help, hints, etc.)

a. was the feedback accurate/appropriate?
b. detailed/not detailed enough?
c. varied/not varied enough?

3) Would you most likely use this program only when the structure is taught in class or could you see yourself using it at a later point again?

4) Additional comments (how did you find the exercises and the "Speed Challenge"?)

5) Have you used Macintoshes before?

List of References

AHMAD, Khurshid et al. Computers, Language Learning and Language Teaching. Cambridge: Cambridge University Press, 1985.

ALESSI, STEPHEN M. and TROLIP, STANLEY R. Computer-Based Instruction: Methods and Development. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1985.

BARKER, PHILIP and Yeates, HARRY. Introducing Computer Assisted Learning. London: Prentice-Hall International, UK Ltd., 1985.

BROWN, C. MARLIN "LIN". Human-Computer Interface Design Guidelines. Norwood, New Jersey: Ablex Publishing Co., 1988.

BURNS, HUGH, PARLETT, JAMES W., and REDFIELD, CAROL LUCKHARDT (eds.). Intelligent Tutoring Systems: Evolutions in Design. Hillsdale, New Jersey: Lawrence Erlbaum Assoc., Inc., 1991.

COLE, PETER and MORGAN, JERRY L. (eds.). Syntax and Semantics, Volume 3, Speech Acts. New York: Academic Press, Inc., 1975.

CRISWELL, ELEANOR L. The Design of Computer-Based Instruction. New York: Macmillan Publishing Co., 1989.

HAMMERLY, HECTOR. An Integrated Theory of Language Teaching and its Practical Consequences. Blaine, Washington: Second Language Publications, 1985.

HAMMERLY, HECTOR. Synthesis in Language Teaching: An Introduction to Languistics. 2nd Edition. Blaine, Washington: Second Language Publications, 1986.

HIGGINS, JOHN. Language, Learners and Computers: Human Intelligence and Artificial Unintelligence. New York: Longman Publishing Group, 1988.

HOLMES, GLYN and KIDD, Marilyn E. "Second Language Learning and Computers", Canadian Modern Language Review, vol. 38, pp. 503-516, 1982.

KRASHEN, STEPHEN. "The Monitor Model for Adult Second Language Performance," pp. 152-61. In Burt, M., Dulay, H., and Finocchiaro, M.

(eds.). Viewpoints on English as a Second Language: Trends in Research and Practice. New York: Regents, 1977.

LAREAU, PAUL and VOCKELL, EDWARD. The Computer in the Foreign Language Curriculum. Santa Cruz, California: Mitchell Publishing, Inc., 1989.

MARSHALL, D.V. CAL/CBT— The Great Debate. Bromley: Chartwell-Bratt, 1988.

PRICE, ROBERT V. Computer-Aided Instruction: A Guide for Authors. Pacific Grove, California: Brooks/Cole Publishing Co., 1991.

ROACH, PETER (ed.). Computing in Linguistics and Phonetics: Introductory Readings. London: Academic Press Limited, 1992.

SHNEIDERMAN, BEN. Designing the User interface: Strategies for Effective Human-Computer Interaction. Reading, Massachusetts: Addison-Wesley Publishing Co. 1987.

SMITH, WM. FLINT (ed.). Modern Media in Foreign Language Education: Theory and Implementation. Chicago: National Textbook Co., 1987.

STEINBERG, ESTHER R. Computer-Assisted Instruction: A Synthesis of Theory, Practice, and Technology. Hillsdale, New Jersey: Lawrence Erlbaum Assoc., Inc., 1991.

TEICHERT, HERMAN U. An Experimental Study Using Modified Individualized Instruction in Beginning College German. UGa., Athens, 1977.

UNDERWOOD, JOHN H. Linguistics, Computers and the Language Teacher: A Communicative Approach. Rowley, Massachusetts: Newbury House Publishers, Inc., 1984.

VENEZKY, RICHARD and OSIN, LUIS. The Intelligent Design of Computer-Assisted Instruction. New York: Longman Publishing Group, 1991.

WENGER, ETIENNE. Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge. Los Altos, California: Morgan Kaufmann Publishers, Inc., 1987.

WYATT, DAVID H. (ed.). Computer-Assisted Language Instruction. Oxford: Pergamon Press Ltd., 1984.