

Web Delivery of Adaptive and Interactive Language Tutoring

Trude Heift¹ and Devlan Nicholson²

¹Linguistics Department and ²Natural Language Laboratory, Simon Fraser University, Burnaby, B.C., Canada V5A1S6, {heift@sfu.ca},{devlan@cs.sfu.ca}

Abstract: This paper discusses the inherent goals and trade-offs involved in the design of an efficient, robust Web tutor within the context of a working, hypermedia framework. The focus is on the *German Tutor*, an Intelligent Language Tutoring System (ILTS) that has been specifically designed for Web delivery. Web-based applications require a high degree of adaptivity and interactivity to address and serve the vast diversity of users in a timely fashion. Our approach emphasizes generality with error-detection algorithms that are not limited to a particular native language user group. Individualization is achieved through a dynamic Student Model that modulates feedback messages and provides remedial tasks suited to learner expertise. System performance and efficiency is enhanced using a modular system within a three-tier architecture. The ILTS has been used with several language classes and results will be reported making reference to the special requirements of efficient, adaptive hypermedia systems.

INTRODUCTION

As Web technologies for adapting existing educational content converge with increased bandwidth, more uniquely Web-based applications are emerging. But while the Internet's convenient access, currency, variety, and integrated multimedia can extend traditional instruction, it also requires a more profound degree of adaptivity and interactivity.

The diversity of users alone presents a design challenge. First of all, the system needs to be general enough to suit a variety of competencies, and backgrounds. For example, Intelligent Language Tutoring Systems (ILTSs) have commonly based their error analysis at least partly on the native language of the student (Schuster, 1986; Catt & Hirst, 1990; Wang & Garigliano, 1995). An on-line system, however, must adopt a more general scheme in order to accommodate international access and cases where the native language of the user might not be known. At the same time, instruction needs to be individualized to address the diverse skill sets of the users. An ILTS that provides individualized instruction, however, requires a fair amount of intelligence built into the system.

System performance and efficiency are also central to interactive learning support systems. Any interaction with a visible delay between the browser and the server defeats the purpose of an interactive system and can, in the worst case, distract from the problem solving activity (Eliot, 1997). While efficiency motivates most server arrangements from form-based CGI interfaces (Brusilovsky, Ritter, & Schwarz, 1997), to Java technology (Warendorf & Tan, 1997; Peylo et.al., 2000), to a combination thereof (Faulhaber & Reinhardt, 1997), we have adopted and will describe a heterogeneous three-tier architecture that integrates multiple servers and several programming languages. A primary Web server interacts with the client machine via CGI scripts and JAVA technology while the intelligent and adaptive components reside on a separate server dedicated to answer processing. This layered design has proven to be efficient and scalable.

Generally, we discuss an ILTS for German which forms the grammar practice component of a Web-based language course. The *German Tutor* has been specifically designed for Web-delivery. Our approach emphasizes adaptivity and interactivity through an intelligent and general but yet individualized system.

Intelligence is imparted by a grammar and a parser that analyzes student input without making any particular assumptions about the user's native language. A student model that maintains learner profiles, displays instructional feedback suited to learner expertise, and suggests remedial exercises ensures an individualized learning environment. Answer processing components are kept separate from learner feedback, user modeling, and the task through a modular design. For example, student exercises are stored in a plain text file which the JAVA

code extracts and displays on a Web page. Instructors can change feedback messages, or, modify exercises, to suit their pedagogical needs, creating a more flexible authoring environment.

In the following, we will describe the system architecture, the error detecting mechanism, and the student model of the *German Tutor*. In all sections we will make reference to design considerations emphasizing efficiency, modularity, generality and individualization. We will also provide results of a system trial and conclude with further research suggestions.

SYSTEM OVERVIEW

The *German Tutor* is an ILTS that forms the grammar component of a comprehensive Web-based course for German. The system contains a grammar and a parser which analyzes sentences from the student and detects grammatical and other errors. The feedback modules of the system correlate the detailed output of the parser with an error-specific feedback message. Currently, there are six exercise types implemented in the system: Dictation, Build a Phrase, Which Word is Different, Word Order Practice, Fill-in-the-Blank, and Build a Sentence. They all differ with respect to the exercise focus and task, and with respect to their use of media and the NLP capabilities of the *German Tutor* (for a detailed description and discussion of the exercise types, see Heift 2001b). The content of the exercises (vocabulary and grammatical structures) is closely related to each chapter of the course.

A Three-tier Architecture

The *German Tutor* relies on a three-tier architecture, illustrated in Figure 1.

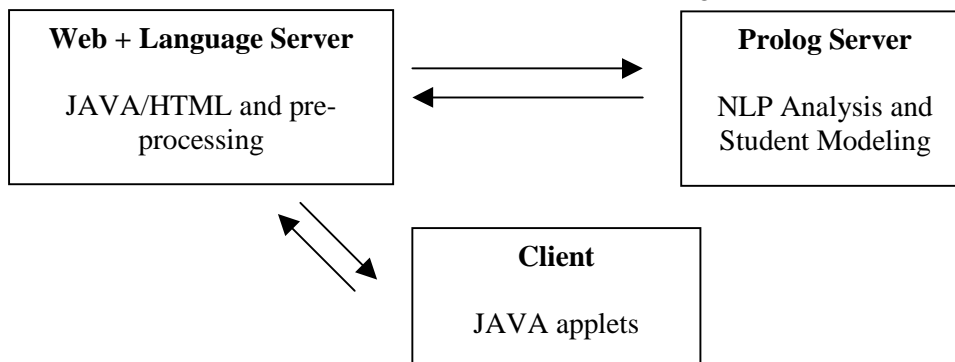


Figure 1: An Interactive Learning Support System

Processing of student input is distributed over two computers for performance reasons. The intelligent and adaptive parts of the answer checking process reside on a dedicated UNIX machine with two CPUs running Prolog while the Java and HTML serving and pre-processing is handled by a Web and Language server.

The Web and Language server handles the interaction between the client and Prolog application. JAVA applets, rather than HTML, have proven to be more suitable in providing the student exercises, while communicating reasonably efficiently with the Prolog server. While JAVA applets require initial loading time, Web pages do not have to be reloaded for each exercise. In addition, navigation within the task is appropriately restricted. Stern (1997:1), for example, points to a problem with some hypermedia systems: "the built-in back buttons of a Web browser allow students to see previous exercises which might not always be desirable in a hypermedia system".

Figure (2) illustrates a sample Web exercise with a student response and system feedback. The student is asked to make a sentence using the words provided [Bianca / to go (perfect) / without / he]. The student input is grammatically incorrect and the system responds with "The auxiliary HAT is wrong".

When the user requests an exercise, the Web browser calls a cgi script on the Web Server passing the module name, the number of exercises, how often to show the summary page, and

the address of the html page that contains the Java exercise applet. All this information is provided by a link on a page already served by the Web Server.

The screenshot shows a web-based language exercise interface. At the top, it says "Guten Tag, John!" in a red font. To the right is a button labeled "Umlaute + ß". Below this, the instruction "Bilden Sie einen Satz mit den folgenden Wörtern." is displayed. A thick horizontal line separates the instruction from the exercise details. Below the line, it says "Übung 3 von 20" and "Bianca / gehen (Perfekt) / ohne / er.". A text input field contains the user's answer: "Bianca hat ohne er gegangen". To the right of the input field are three buttons: "Prüfen", "Lösung", and "Weiter >>". Below the input field, a red error message reads: "Hier stimmt das Hilfsverb HAT nicht."

Figure 2: A Sample Exercise

The cgi script, written in Perl, generates a new html page from the information given by the Web browser by replacing various keywords on the page including a placeholder for an exercise applet.

The Web browser updates and requests the new applet. Once the Java code is received, the applet starts.

The JAVA applet calls the Language Server asking for exercise data, passing the user's name, module, and number of exercises to retrieve. The Language Server verifies that the user and the module exist and returns the requested exercise data. The Language Server is also written in Java.

The second task of the Web server is to handle pre-processing of student input as discussed in the following section.

A Modular Approach to Answer Processing

Figure (3) illustrates the flow of a sentence through the various answer checking modules. All modules except the spell and grammar check reside on the Web server and are implemented in JAVA. These JAVA components, however, share preliminary information from the Prolog server regarding the expected constraints of the exercise, which is provided during the spell check stage detailed below.

The system is organized in such a way that if any error checking module detects an error, further processing is blocked. As a result, only one error at a time will be displayed to the learner. Studies have shown (van der Linden, 1993) that displaying more than one feedback message at a time makes the correction process too complex for the student. From a computational point of view, interrupting the error processing mechanism also assists in distributing the queue for each module in case of multiple, simultaneous users.

For each exercise, the Java applet creates a new session and displays the task to the user. The user enters an answer and asks the applet to check the answer. The Java applet calls the Language Server giving it the session data.

The Language Server performs the following checks in order until an error is flagged or the sentence is judged to be correct and the Language Server returns. Before returning to the applet, the Language Server will update the user's student model based on what happens during the checks.

String Match

When the user input is first received by the server, the system performs a string match comparing the user's answer to pre-stored answers. Each pre-stored answer is accompanied by a previously computed analysis for student model updates. Storing multiple answers allows for

freer student input and, at an introductory level, involves little extra work. At an advanced level where exercises become even less constrained, alternatives to pre-storing correct answers can be considered (see Gerbault, 1999).

If the student input matches any of the stored answers, then the user's student model is updated and the Language Server returns the session back to the Java applet. A previous study (Heift & Nicholson, 2000a) has shown that approximately 40% of all answers submitted are initially correct. In these instances, the complete error checking mechanism is unnecessary if it can be quickly determined that the sentence is correct. If the string match fails, the sentence is passed to the Punctuation Check.

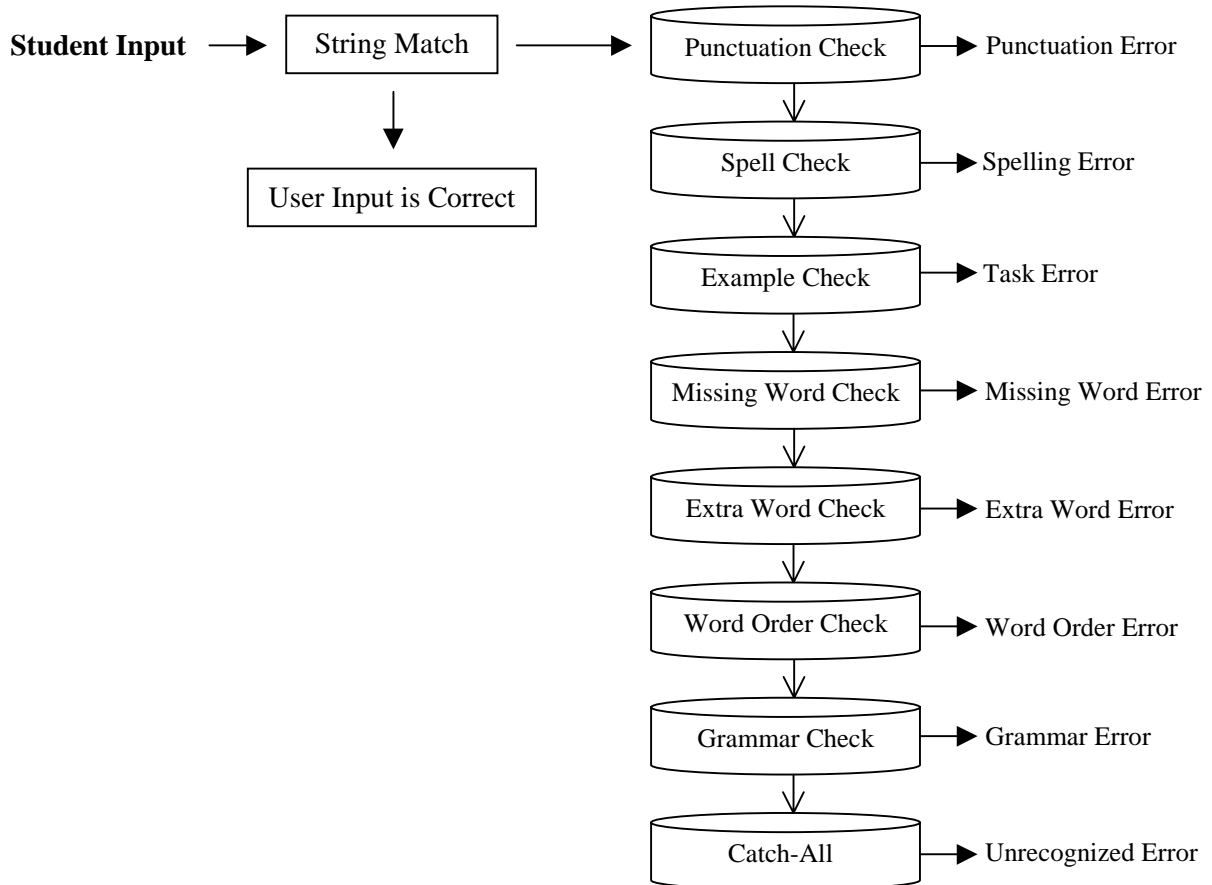


Figure 3: Error Sieve

Punctuation Check

Here the Language Server checks for punctuation. If no punctuation error is found the Language Server proceeds to the next error checking module.

Spell Check

During the spell check, the Language Server calls the Prolog Server with each word in the user's answer. It connects using sockets, so there is no restriction on what machine the Prolog Server may be running. For reasons of efficiency, however, we are using a dedicated, two CPU machine running Prolog. The Prolog Server then calls Ispell with the word to check. We are using Ispell Version 3.1.2 with our own dictionary containing approximately 6000 words. Ispell returns whether the word is spelled correctly, and possible alternatives if it is not.

The Prolog Server then attempts to determine the lemma, or base form, of the word. The results and the base form are then returned to the Language Server. The Language Server continues checking the input sentence unless the user's answer contains a spelling mistake or the Prolog Server was unable to determine base forms for the words.

Example Check

The Example Check compares the supplied baseforms with those required by the current exercise. If necessary, it calls the Prolog Server to determine them. The system reports an error if one of the base forms in the example is missing in the user's answer.

Missing Word Check

During the Missing Word Check, the Language Server calls the Prolog Server which in turn calls the Most Likely Answer (MLA) Analyzer to determine which of the possible answers is most likely intended by the user. The MLA Analyzer is an application written in Perl and C and based on a genetic algorithm (see Huang & Miller, 1991). The MLA analyzer returns the most likely answer as well as word order information. The Prolog Server passes this information to the Language Server. The Language Server continues checking unless one of the base forms in the MLA is missing in the user's answer.

Extra Word Check

Here the Language Server checks for base forms in the user's answer that are not in the MLA. The system reports an error if extraneous words are found in the user input.

Word Order Check

During the Word Order Check, the Language Server determines whether the base forms in the user's answer are in order. The Language Server passes the input sentence to the Grammar Check unless the word order is incorrect.

Errors in omission, insertion and word order are commonly handled by the grammar in other systems (Covington & Weinrich, 1991; Schwind, 1995). Our initial testing, however, showed that system performance is higher if these error types are treated externally. This avoids the exponential cost of each additional rule within a declarative grammar.

Grammar Check

The Grammar Check is the most elaborate of the tests. Here, the Language Server calls the Prolog Server with the user's answer. The Prolog Server passes the answer through a grammar (and various other components) and then returns to the Language Server update information for the student model and feedback messages for the user. The sentence is passed to the final module unless a grammar error has been detected.

Catch-All

The Catch-all is required for completeness. If a sentence is initially determined to contain an error but none of the error checking modules can pinpoint its source then a generic message is sent. The mechanism is rarely invoked: for the study reported later in the article, for example, the Catch-all never applied.

Result

When the Java applet gets the session back from the Language Server, it shows the results to the user. If the exercise was correct, the user may move on to the next exercise. Otherwise, the user may resubmit an answer.

Each answer checking module is responsible for different error classes and operates independently. For example, instructors who do not want sentences checked for punctuation can disable the punctuation check without affecting the remaining code. The system also separates the answer checking mechanisms from the student tasks. Student exercises are extracted from a separate text file. This modular design allows for convenient authoring. Instructors can design their own exercises without any specialized knowledge of the system's sophisticated technology. Even more importantly, the modular design results in higher accuracy because the answer checking mechanism does not rely solely on the grammar and the parser. Specialized modules use whatever techniques are most appropriate and efficient for each error class. Moreover, the additional error checking modules make our system less dependent on a

particular native language, for reasons discussed in the following section.

GENERALITY

ILTSs provide a practice environment for second language learners who, due to their language competence level, frequently key in incorrect sentences. Over the past two decades, the major challenge for Intelligent Language Tutors has been the analysis of ill-formed student input in language learning environments that go beyond multiple choice questions (Weischedel, 1983; Schwind, 1995; Schneider & McCoy, 1998). A common approach to achieving good error coverage contrasts the student's native language with the target language, determines the most likely errors, and builds the system's analysis around these common errors. For example, Yang & Akahori (1997, 1999) conducted a study investigating the error types for the passive voice in Japanese and implemented the required rules for parsing ill-formed sentences accordingly. However, in a Web-based system any presumption of a particular student profile is severely restrictive given the international nature of the Internet.

Error analysis in the *German Tutor* is not restricted to a particular native language. Firstly, the entire interface including the feedback messages are kept in the target language so as not to limit the system to a particular user group. Secondly, and more importantly, we employ different analyses for errors that are finite versus those errors which are less predictable and unlimited in scope. For example, in German there are four grammatical cases: nominative, accusative, dative and genitive. Similarly, there are three genders: feminine, neuter, and masculine and two numbers: singular and plural. Any of these errors are processed efficiently by the grammar and the parser because if the student makes an incorrect choice the answer will contain one of the few remaining possibilities. In our approach, the grammar parses any of the alternative forms and keeps track of whether or not the student's choice was correct (see the NLP analysis in the next section).

In contrast to these finite errors, there are error classes which are less predictable. For example, given a sentence any word may be inserted, missing or out of order albeit due to native language interference or even inattentiveness. Anticipating these errors within a declarative grammar would certainly make the system inefficient. For this reason, we treat errors in omission, insertion and word order outside the grammar. Each error class has its own language-independent algorithm which detects the errors and operates independently within the modular design described above.

Our approach has two main advantages. First of all, the system is not limited to a particular native language without sacrificing efficiency. Second, the system is highly accurate because errors in omission, insertion or word order which commonly contribute to sentence ambiguity are not handled by the grammar and the parser. By the time the sentence is sent to the grammar these errors have been detected by their respective modules.

In an ILTS, there is commonly a trade-off between system performance and depth of error analysis. Generally, the less sophisticated the error analysis the more efficient the system. For example, a system which employs a simple string-matching algorithm is faster than a system which keeps track of specific errors and makes pedagogical distinctions accordingly. However, student feedback is also less detailed in the former system because it lacks the deep error analysis that is required for generating error-specific feedback.

In our approach, we opted for efficiency for errors in insertion, omission and word order due to the requirements of a Web-based system. This choice, however, did not result in a loss of feedback specificity. For errors in insertion and omission, the student will be given the particular word that is extraneous or missing. Similarly, for errors in word order we inform the user which word is out of order.

In the following we will discuss the Natural Language Processing (NLP) component of the *German Tutor*.

NLP ANALYSIS

The NLP component of the system consists of a grammar and a parser. The grammar formalism

used is derived from Head-driven Phrase Structure Grammar (Pollard & Sag 1987, 1994) which is well suited to the task at hand (for a detailed discussion of other grammar formalisms and NLP systems, see Heift, 1998). HPSG is a lexicalist approach in which linguistic information is presented as feature/value matrices. For example, Figure (4a) illustrates a minimal lexical entry for *geht*. The subcategorization list of the verb, notated with the feature *subj*, specifies that *geht* takes a subject which is minimally specified as a singular noun. Rules of grammar specify how words and phrases are combined into larger units according to the subcategorization list. In addition, other principles govern how information such as the head features, which inter alia determine the grammatical category of a constituent, is inherited.

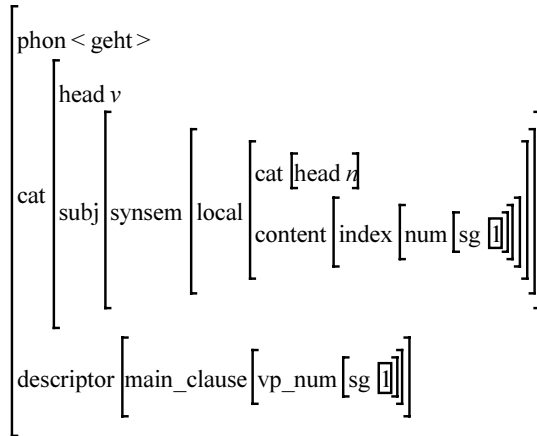


Figure 4a: Recording Number Features for *geht*

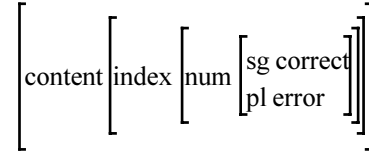


Figure 4b: Number Features for a Singular Noun

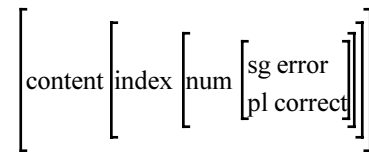


Figure 4c: Number Features for a Plural Noun

Unification-based grammars place an important restriction on unification, namely that two categories A and B fail to unify if they contain mutually inconsistent information (Gazdar & Pullum, 1985). If unification fails, however, we receive no details of the analysis. To overcome this restriction, we relax some constraints. Number agreement, for example, records whether or not the subject of *geht* is in the singular. To achieve this, the noun is no longer marked as [num sg], but instead a new path num|sg terminates with the values error or correct. For example, for a singular noun phrase, the value of the path num|sg is correct, while it is error for a plural noun phrase. The two partial lexical entries are given in Figure 4(b) and Figure 4(c), respectively.

The verb *geht* records the value of *sg* from its subject (Figure 4a). If the value of the path num|sg is correct, the subject is in the singular. In case of a plural noun, *geht* records the value error for number agreement, but unification succeeds.

The goal of the parser and the grammar is the generation of phrase descriptors representing the description of the phrase that the parser builds up. The type *descriptor* is set-valued and is initially underspecified in each lexical entry. During parsing, the values of the features of *descriptor* are specified. For example, one of the members of *descriptor*, *vp_num* in Figure (4a), records the number agreement of subject-verb in a main-clause. Its value is inherited from the *sg* feature specified in the verb *geht*. Ultimately, the descriptor records whether the sentence is grammatical, what errors were made and, eventually, is associated with a feedback message. The algorithm has been applied to all syntactic agreement phenomena and such semantic restrictions as those on the subjects of verbs.

In the next section, we will describe the student model of the *German Tutor*.

INDIVIDUALIZATION OF THE LEARNING PROCESS

A number of user modeling techniques have been employed in ILTSs over the past decade (Catt & Hirst, 1990; Kurup, Greer, and McCalla, 1992; Bull, 1994; Bailin, 1990, 1995; Kang & Maciejewski, 2000) but the challenging task in hypermedia systems lies in overcoming the statelessness of the HTTP protocol for identifying students and maintaining user models adaptive to the individual (Stern, 1997). A number of solutions have been proposed ranging

from cookies, structured URLs, hidden fields to login screens (Stern, 1997; Eliot, 1997; Yang & Akahori, 1997). The choice depends primarily on the application.

The *German Tutor* keeps a database of users, each entry established by an initial login. The student login and password is sufficient for the design of our system since students do not navigate through different HTML pages during grammar tasks, but access a consistent applet. However, student identification is required to support the Student Model's main functions:

First, for each student the Student Model keeps score across a number of error types, or, nodes such as grammar, vocabulary, punctuation, etc. The grammar node is broken down into more fine-grained categories. For example, it will contain detailed information on the student's performance on subject-verb agreement, case assignment, etc. as obtained from the phrase descriptors discussed in the NLP section. At the end of each exercise set, students receive detailed information on the errors they have made as given in Figure (5).

Guten Tag, John!

Bilden Sie einen Satz mit den folgenden Wörtern.

Hier sind Ihre Ergebnisse (10 Übungen von 10): Weiter >>

Rechtschreibung : 1 Fehler

Subjekt-Verb Übereinstimmung : 10 Fehler

**Sie sollten vielleicht Subjekt-Verb Übereinstimmung noch etwas üben.
Bitte klicken Sie auf WEITER.**

Schicken Sie Ihre Ergebnisse an Ihren Professor.

E-mail Adresse von Ihrem Professor: Schicken

Figure 5: Student Summary Page

The summary page in Figure (5) states that the student *John* made one spelling mistake and ten errors in subject-verb agreement. Due to the number of errors, the system suggests further exercises on subject-verb agreement. The student will receive an individually tailored set of remedial exercises addressing the mistakes he made during previous practice. The results can also be sent to the instructor.

A second function of the Student Model is to modulate feedback messages suited to learner expertise. Student performance history determines the specificity of the feedback displayed to the learner (see also Brusilovsky et al., 1996). The central idea is that within a framework of guided discovery an expert requires less detailed feedback than a beginner learner (Fischer & Mandl, 1988; Elsom-Cook, 1988). For each grammar node (dative case, two-way prepositions, etc.), the system creates three categories of instructional feedback corresponding to three learning levels: beginner, intermediate, and advanced. Provided with three categories of feedback, the system selects an error response suited to the student's expertise.

The Student Model maintains the learner model update by keeping score for each node. Depending on student input, the score for each node will go up or down. The amount by which the scores of each node are adjusted is specified in a master file and may be weighted to reflect different pedagogical purposes, such as emphasis of an exercise, importance of an error, etc. The messages are then selected according to the current learner level at each error node. Initially, all students start out at the intermediate level although pre-testing each student would be a feasible alternative. However, due to the detailed scoring, the system nodes adjust quickly.

The main strength of the Student Model in the *German Tutor* is that a single erroneous message will not drastically change the overall assessment of the student. The phrase descriptors collect errors that indicate precisely which grammatical violations occurred, allowing for a fine-grained assessment of student competency. In consequence, a student can be at a different level for each given grammar constraint reflecting her performance of each particular grammatical

skill. This subtlety of evaluation is desirable in a language teaching environment because as the student progresses through a language course a single measure is not sufficient to capture the knowledge attained and to distinguish among learners. The Student Model aids in directing each student toward error-specific and individualized remediation.

In the final section, we will discuss the results of a system trial making reference to our design goals of generality, individualization and efficiency.

SYSTEM TRIAL

The system was used by 33 students during three one-hour class sessions in the spring semester 2000. In addition, students were able to use the *German Tutor* during the entire semester from their home computer. During the three one-hour sessions students worked on 6 chapters with a total of 120 exercises. The system kept a detailed log of user interactions (Heift & Nicholson, 2000b). In addition, students filled out a questionnaire.

A total of 4405 sentences were analyzed by the system. The following is a sample of the data log for each user and sentence:

```
(1) 00/02/01 13:48:34 D1 bas german/beginner/chapter_three
(2) ich / in / Rom / wohnen
(3) Ich wohnen in Rom.
(4) Achtung! Die Verbform von WOHNEN und das Subjekt ICH passen nicht zusammen. ICH ist
singular.
Watch out! The verb WOHNEN and the subject ICH do not agree. ICH is singular.
(5) ps=1 ss=38 es=1 ms=32 xs=1 os=1 gf=193
(6) tutor_spelling dec 18:03 tutor_example dec 12:03
tutor_missing dec 15:03 tutor_extra dec 12:03
tutor_order dec 15:03 tutor_grammar inc 15:03
(7) subject dec 12:03
vp_number inc 15:03 vp_person inc 15:03
pp_accdat dec 12:03 pp_accdat_noun dec 12:03
```

Line (1) of the data sample shows the date, time, user (D1), exercise type (build a sentence) and the particular chapter (chapter 3). Line (2) displays the exercise followed by the student answer on line (3). Line (4) shows the system feedback which allows us to verify that the system identified the error correctly and displayed the appropriate feedback message. Line (5) records the processing time of each error checking module in milliseconds (ps = Punctuation Check, ss = Spell Check, es = Example Check, ms = Missing Word Check, xs = Extra Word Check, ws = Word Order Check, gf = Grammar Check). The second letter in each case indicates whether the answer succeeded ('s') or failed ('f'). Thus the data above show that the answer checking mechanism terminated with the grammar check because the parser and grammar found a mistake in the sentence. The total processing time of the student's input was 267 ms with the spell checker, missing word check, and the grammar check taking the most time.¹

Line (6) and (7) provide data on the specific error types required for the Student Model but which, for the purpose of this paper, will not be explained in great length. Briefly, line (6) shows the node and the current score. For example, the *tut_spelling* node refers to the spellchecker module. 'dec' indicates that a value was decremented since the sentence did not contain any spelling errors. As in golf, the lower scores are better. The current spelling score for the student is 18 and the spellchecker node was activated three times for student D1 during total practice time. The possible values range from 0 – 30, 0-10 for the advanced learner, 11-20 for the intermediate, and 21-30 for the beginner. In contrast to the spelling node, the *tut_grammar* node indicates an increment since the sentence contained a grammar error. The current score for the grammar node is 15 and the node has also been activated three times.

Line (7) provides a detailed analysis of the grammar nodes which were activated during the

¹ The time log does not indicate the actual processing time, but rather local network response time, i.e., the time between when a sentence enters and leaves each module. The time log does not, however, include the Internet response time.

error checking process. For example, the prepositional phrase *in Rom* in the student input was correct. Thus the nodes 'pp_accdat' and 'pp_accdat_noun' indicate a decrement. However, the sentence contained an error with subject-verb agreement. Therefore the nodes 'vp_number' and 'vp_person' show an increment.

Leaving off further explanation of lines (6) and (7), we will now examine data which relates to accuracy and performance of the system.

System Accuracy and Generality

Out of the 33 students who used the system during the spring semester, 9 students were non-native speakers of English with a variety of native languages: Japanese, Mandarin, Cantonese, Czech, Polish, Punjabi and French.

Of the 4405 sentences processed by the system, 1791 were submitted correctly on the first try. For the remaining 2614 sentences, the system performed an accurate error analysis, although in some instances, the feedback was not as detailed as we had hoped. In these cases, the student error was caught by the Example Check where feedback tends to be less specific than if analyzed by the grammar and the parser. Subsequent changes to the lemmas of some of the lexical entries led to results which are consistent with intended design.

It is worth noting that, after extensive beta-testing, the performance of the system rivals a careful human analysis. Moreover, the data confirm that the error analysis of student input is not tied to a particular native language. The 4405 sentences which were processed by the system were analyzed correctly despite the fact that almost one third of the participants were non-native speakers of English. Moreover, examples (1) - (3) illustrate the unpredictability of some errors which occurred during the study sessions.

- | | |
|--|--|
| (1a) Task: wir / kaufen / Klaus / (indef. Artikel) / Krawatte | |
| (1b) Student Input: *Wir kaufen Klaus einer der Krawatte. | <i>*We are buying a the tie for Klaus.</i> |
| (1c) Correct Response: Wir kaufen Klaus eine Krawatte. | <i>We are buying a tie for Klaus.</i> |
| (1d) System Response: Überflüssige Wörter: DER | <i>Extra Words: DER</i> |
| (2a) Task: Gehen (Perfekt) / sie (Singular) / in / (def. Artikel) / Stadt? | |
| (2b) Student Input: *Hat sie in der | <i>*Has she in the</i> |
| (2c) Correct Answer: Ist sie in die Stadt gegangen? | <i>Did she go downtown?</i> |
| (2d) System Response: Fehlende Wörter: GEHEN, STADT | <i>Missing words: GEHEN, STADT</i> |
| (3a) Task: heute / ich / nicht / frühstücken (Perfekt). | |
| (3b) Student Input: * Heute ich habe nicht gefrühstückt. | <i>Today I have not had breakfast.</i> |
| (3c) Correct Answer: Heute habe ich nicht gefrühstückt. | <i>Today I have not had breakfast.</i> |
| (3d) System Response: Vorsicht, die Wortstellung ist falsch. | <i>Watch out! The word order is wrong.</i> |

In example (1) the student inserted an extra article (*der*). In example (2) the student left out the main verb and the noun (*gehen, Stadt*). In example (3) the student did not place the inflected verb in second position, a mistake commonly made by English speakers due to native language interference. For all three instances, the grammar and the parser would require additional rules to parse these errors. These rules, however, would also need to account for the rather unpredictable errors given in examples (1) and (2). Our generic algorithms successfully process these errors without sacrificing system efficiency.

The high accuracy of our system is due to at least three factors. First, the language tasks, although allowing for freer input, nonetheless constrain the domain. The system displays tasks in which students select from a given pool of vocabulary and grammatical structures. Second, students' language proficiency is still limited at an introductory level and thus neither the grammatical constructions nor the vocabulary are highly complex. Third, pre-processing the sentence for spelling, omission, insertion and word order errors assists the grammar and the parser by eliminating much of the ambiguity of erroneous natural language.

Unlike the grammar and the parser, the algorithms for errors in omission, insertion and word order, while general, require all possible answers stored with the exercise. Thus the current system is not designed for free input. A free input system, however, would defeat the purpose of our initial design goals. We were aiming at a highly accurate, meaningful practice environment

for second language learners. Our system takes advantage of the constrained domain given by a beginner learner level in achieving these goals. In an open domain, any NLP system tends to become less accurate.

Individualization

When analyzing the data with respect to individualized instruction, we were interested in the types of errors that occurred during practice and their distribution with respect to the three learner levels: beginner, intermediate and advanced. Table (1) shows the error break-down by answer checking module and learner level.

	Beginner	Intermediate	Advanced	TOTAL	%
Punctuation Check	2	41	5	48	2.5%
Example Check	3	126	21	150	7.9%
Spelling Check	72	278	6	356	18.7%
Missing Word Check	11	37	18	66	3.5%
Extra Word Check	11	19	11	41	2.1%
Word Order Check	10	14	10	34	1.8%
Grammar Check	228	874	109	1211	63.5%
Check-All	0	0	0	0	0%
TOTAL	337	1389	180	1906	100%

Table 1: Error Break-down

Table (1) indicates that roughly two thirds of the errors were analyzed by the grammar and the parser (63.5%). Spelling errors were the second most frequent errors during practice (18.7%).

Students were most often at the intermediate level, which is not surprising since each student is initially placed at the intermediate level. It is interesting to note, however, that although beginner students committed more spelling and grammar errors than advanced students, they do not appear especially more prone to committing other types of errors.

Our data emphasize the importance of an adaptive language learning system. Approximately 37% of the time, students either required more elaborate feedback suited to the beginner learner, or, in the case of the advanced learner, less detailed feedback was sufficient to correct the errors. Moreover, and although not illustrated in Table (1), ten students or 30.3% of all participants received remedial exercises for at least one of the six chapters.

Previous studies which investigated the pedagogical impact of the system (Heift, 2002, 2001a) indicated that depending on their performance level students apply different learner strategies during practice and error correction. These results further justify the need for an individualized system. It is the goal of future studies to examine the overall impact of modulated feedback.

Efficiency

When we first completed the system, we experienced problems with multiple users and discovered heavy time delays which were due to unrelated user traffic on the Web server. Moving our Web and Language server to a dedicated server eliminated most of the problem. Since the initial testing phase of the system, we also implemented the initial string match discussed earlier and made improvements to the lexical look-up.

The advantage of the initial string match is that we can quickly determine when a sentence is correct. It thus supports our modular design which, besides being motivated by pedagogical reasons, assists in distributing the queue for each module in case of multiple, simultaneous users.

System modifications to the lexical look-up improved the speed of the spell checker and the grammar. In (1a) and (1b) we display the processing times for each answer-processing module for the incorrect sentence **Ich darf keine Bier trinken (I am not allowed to drink any beer)* before and after the system modifications, respectively.

(1a) ps=3	ss=914	es=1	ms=75	xs=1	ws=1	gf=1366
(1b) ps=1	ss=311	es=0	ms=72	xs=0	ws=0	gf=422

The data in (1a) and (1b) indicate that after our modifications to the lexical look-up, total processing time for the incorrect sentence **Ich darf keine Bier trinken* improved from 2361ms to 806ms.

We believe that the speed of our ILTS is reasonable for an interactive Web-based tutor. This is supported by 94% of the study participants who indicated in the questionnaire that they felt that the system was fast enough. Furthermore and as indicated in Table (1), only 63.5% of all sentences made it all the way through the grammar and the parser. Due to our modular design, however, the remaining sentences were caught in earlier modules and thus required less processing time.

CONCLUSION AND FURTHER RESEARCH

Web-based language tutors require a high degree of adaptivity and interactivity to address and serve the large variety of users in a timely fashion. In this paper we discussed an Intelligent Language Tutor which emphasizes efficiency, modularity, generality, and individualization as essential design criteria for an adaptive and interactive Web-based tutor. Our analysis of student input is general in that it is native language independent and thus not constrained to a particular user group. At the same time, the learning process is individualized through a dynamic Student Model. The system is highly modular allowing for convenient authoring, as well as ease of maintenance and adaptability. Efficiency is achieved with a three layer architecture: a Web server interacts with the client machine via CGI scripts and JAVA technology and the intelligent and adaptive components reside on a separate server dedicated to answer processing.

The World Wide Web has provided convenient access to ILTSs which, due to the processing requirements of the grammar and the parser would hardly be feasible in any other format. In a Web-based environment we can control the speed and memory requirements on our server instead of putting the demand on the user. As a result, any student with Internet access can use the system.

While we are satisfied with the overall efficiency of the *German Tutor*, we would like to make the system even more adaptive. For example, our Student Model could also be used to select exercises suited to learner expertise which in turn would require careful sorting of exercises by difficulty level. We also would like to exploit the Hypermedia environment to a larger extent. For instance, in addition to modulating feedback messages and providing remedial exercises we are planning to extend the system with reference tools. Web pages can be cross-referenced by grammar topic and native language and suggested to the student from within our system when needed.

ACKNOWLEDGEMENTS

We would like to thank the editors and anonymous reviewers for their insightful comments and suggestions.

This research was supported by SSHRC Grant 632209.

REFERENCES

- Bailin, A. (1995). AI and Language Learning. *Intelligent Language Tutors: Theory Shaping Technology*. Holland, M. V., Kaplan, J.D., Sama, M.R., eds. Mahwah, New Jersey: Lawrence Erlbaum Associates, Inc.: 327-343.
- Bailin, A. (1990). Skills in Context and Student Modeling. *CALICO Journal*, 8: 7-22.
- Brusilovsky, P. (1998). Adaptive Educational Systems on the World-Wide-Web: A Review of Available Technologies. *Web-based ITS at ITS '98*, 4th International Conference in Intelligent Tutoring Systems, San Antonio, Texas.
<http://www-aml.cs.umass.edu/~stern/webits/itsworkshop/>
- Brusilovsky, P., Ritter, S., and Schwarz, E. (1997). Distributed intelligent tutoring on the Web.

- In: B. du Boulay and R. Mizoguchi (eds.). *Proceedings of AIED '97*, Amsterdam: IOS: 482-489.
- Brusilovsky, P., Ritter, S., and Weber, G. (1996). ELM-ART: An intelligent tutoring system on the World Wide Web. In: Frasson, C., Gauthier, G. and Lesgold, A., eds. *Lecture Notes in Computer Science*. Springer Verlag: 261-269.
- Bull, S. (1994). Student Modelling for Second Language Acquisition. *Computers and Education*, 23 (1-2): 13-20.
- Carpenter, B., and Penn, Gerald. (1994). *The Attribute Logic Engine: User's Guide, Version 2.0*. Computational Linguistics Program, Carnegie Mellon University, Pittsburgh.
- Catt, M., and Hirst, G. (1990). An Intelligent CALI System for Grammatical Error Diagnosis. *Computer Assisted Language Learning*, 3: 3-27.
- Covington, M.A., and Weinrich, K.B. (1991). Unification-Based Diagnosis of Language Learners' Syntax Errors. *Literary and Linguistic Computing*, 6 (3): 149-154.
- Eliot, C. (1997). Implementing Web-Based Intelligent Tutors. *Proceedings of the Workshop Adaptive Systems and User Modeling on the World Wide Web*, 6th International Conference on User Modeling, Chia Laguna, Sardinia.
http://www.contrib.andrew.cmu.edu/~plb/UM97_workshop/Eliot.html
- Elsom-Cook, M. (1988). Guided Discovery Tutoring and Bounded User Modelling. *Artificial Intelligence and Human Learning*. Self, J., ed. Bristol: J. W. Arrowsmith Ltd.: 165-178.
- Faulhaber, S. and Reinhardt, B. (1997). D3-WWW-Trainer. Entwicklung einer Oberfläche für die Netzanwendung. In: *Workshop der GI: Intelligente Lehr/Lernsysteme September 97 in Duisburg, Interner Bericht der TU München TUM-I9736*, 31-40, August 1997.
- Fischer, P. M., and Mandl, H. (1988). Improvement of the Acquisition of Knowledge by Informing Feedback. *Learning Issues for Intelligent Tutoring Systems*. Mandl, H. and Lesgold, A., eds. Springer Verlag: 187-241.
- Frasson, C., Gauthier, G., and McCalla, G.I., eds. (1992). *Intelligent Tutoring Systems*. Lecture Notes in Computer Science. Springer Verlag.
- Gazdar, G., and Pullum, G. (1985). Computationally Relevant Properties of Natural Languages and their Grammars. *New Generation Computing*. 3: 273-306.
- Gerbault, J. (1999). Towards an Analysis of Answers to Open-Ended Questions in Computer-Assisted Language Learning. In: Lajoie, S. and Martial, V., (eds). *Proceedings of AIED '99*, Amsterdam: IOS: 686-689.
- Heift, T. (2002). Learner Control and Error Correction in ICALL: Browsers, Peekers and Adamants. *CALICO Journal*, Volume 19 (3), in press.
- Heift, T. (2001a). Error-Specific and Individualized Feedback in a Web-based Language Tutoring System: *Do They Read It? ReCALL Journal*, Volume 13 (1): 129-142.
- Heift, T. (2001b). "Intelligent Language Tutoring Systems for Grammar Practice". *Zeitschrift für Interkulturellen Fremdsprachenunterricht* [Online], 6(2), 15 pp. Available at: <http://www.ualberta.ca/~german/ejournal/heift2.htm>
- Heift, T. (1998). *Designed Intelligence: A Language Teacher Model*. Unpublished Ph.D. Dissertation, Simon Fraser University.
- Heift, T. & Nicholson, D. S. (2000a). Theoretical and Practical Considerations for Web-based Intelligent Language Tutoring Systems. In Gauthier,G., Frasson,C., VanLehn, K., (eds). *Intelligent Tutoring Systems*. 5th International Conference, ITS 2000, Montreal, Canada, June 2000: 354-62.
- Heift, T. & Nicholson, D. S. (2000b). Enhanced Server Logs for Intelligent, Adaptive Web-based Systems. *Proceedings of the Workshop on Adaptive and Intelligent Web-based Educational Systems*, ITS' 2000, Montreal, Canada: 23-28.
- Holland, M. V., Kaplan, J.D., and Sama, M.R., eds. (1995a). *Intelligent Language Tutors: Theory Shaping Technology*. Mahwah, New Jersey: Lawrence Erlbaum Associates, Inc.
- Huang, X. and Miller, W. (1991). *Advanced Applied Mathematics*, (12): 337-357.
- Kang, Y-S. and Maciejewski, A.A. (2000). A Student Model of Technical Japanese Reading Proficiency for an Intelligent Tutoring System. *CALICO Journal*, Volume 18 (1):9-40.
- Kurup, M., Greer, J. E., and McCalla, G. (1992). The Faulty Article Tutor. *Intelligent Tutoring Systems*. Frasson, C., Gauthier, G., McCalla G.I, eds. Lecture Notes in Computer Science.

Springer Verlag: 84-91.

- Liou, H.-C. (1991). Development of an English Grammar Checker: A Progress Report. *CALICO Journal*, 9: 27-56.
- Peylo, C., Thelen, T., Rollinger, C., Gust, H. (2000). A Web-based Intelligent Educational System for PROLOG. *Proceedings of the Workshop on Adaptive and Intelligent Web-based Educational Systems*, ITS' 2000, Montreal, Canada: 62-68.
- Schneider, D. and McCoy, K. (1998). Recognizing Syntactic Errors in the Writing of Second Language Learners. *Proceedings of the 17th International Conference on Computational Linguistics*.
- Schuster, E. (1986). The Role of Native Grammars in Correcting Errors in Second Language Learning. *Computational Intelligence*, 2 (2): 93-98.
- Self, J., ed. (1988). *Artificial Intelligence and Human Learning*. London, New York: Chapman and Hall Ltd.
- Stern, M. (1997). The Difficulties in Web-based Tutoring, and Some Possible Solutions. *Proceedings of the Workshop Intelligent Educational Systems on the World Wide Web*, 8th Conference of the AIED-Society, Kobe, Japan.
http://www.contrib.andrew.cmu.edu/~plb/AIED97_workshop/Stern.html
- Van der Linden, E. (1993). Does Feedback Enhance Computer-Assisted Language Learning. *Computers & Education*, 21 (1-2): 61-65.
- Wang, Y., and Garigliano, R. (1995). Empirical Studies and Intelligent Language Tutoring. *Instructional Science*, 23: 47-64.
- Warendorf, K. and Tan, C. (1997). ADIS – An animated data structure intelligent tutoring system or Putting an interactive tutor on the WWW. In: Brusilovsky, P., Nakabayashi, K. and Ritter, S. eds. *Proceedings of the Workshop Intelligent Educational Systems on the World Wide Web*, 8th Conference of the AIED-Society, Kobe, Japan.
http://www.contrib.andrew.cmu.edu/~plb/AIED97_workshop/Warendorf/Warendorf.html
- Weischedel, R.M. and Sondheimer, N.J. (1983). Meta-Rules as a Basis for Processing Ill-Formed Input. *American Journal of Computational Linguistics*, Special Issue on Ill-Formed Input, 9 (3-4): 161-177.
- Yang, J. and Akahori, K. (1999). An Evaluation of Japanese CALL Systems on the WWW. Comparing a Freely Input Approach with Multiple Selection. *Computer Assisted Language Learning* 12 (1): pp. 59-79.
- Yang, J. and Akahori, K. (1997). Development of Computer Assisted Language Learning System for Japanese Writing Using Natural Language Processing Techniques: A Study on Passive Voice. *Proceedings of the Workshop Intelligent Educational Systems on the World Wide Web*, 8th Conference of the AIED-Society, Kobe, Japan.
http://www.contrib.andrew.cmu.edu/~plb/AIED97_workshop/Yang/Yang.html