Learning Linked Lists: Experiments with the iList System

Davide Fossati[1], Barbara Di Eugenio[1], Christopher Brown[2], and Stellan Ohlsson[3]

[1] Department of Computer Science, University of Illinois, Chicago, IL, USA
[2] Department of Computer Science, U.S. Naval Academy, Annapolis, MD, USA
[3] Department of Psychology, University of Illinois, Chicago, IL, USA
dfossa1@uic.edu, bdieugen@cs.uic.edu, wcbrown@usna.edu, stellan@uic.edu

Abstract. This paper presents the first experiments with an Intelligent Tutoring System in the domain of linked lists, a fundamental topic in Computer Science. The system has been deployed in an introductory college-level Computer Science class, and engendered significant learning gains. A constraint-based approach has been adopted in the design and implementation of the system. We describe the system architecture, its current functionalities, and the future directions of its development.

1 Introduction

In this paper, we present the first version of iList, an Intelligent Tutoring System (ITS) in the domain of basic data structures and algorithms in Computer Science (CS), and its evaluation. Among the innovative features of our work are: the domain itself, and specifically, our focus on linked lists, due to pedagogical tenets for CS; the choice of constraint-based modeling as the basis for our ITS; and the structure of our graphical interface, itself partly due to the pedagogy of CS. This work is situated within our larger research program, whose main goal is, similarly to others [1–4], to better understand why human tutoring is effective, and to discover computational models of effective tutoring that can be implemented in ITSs. We are developing a second version of the ITS we present here, based on our data collection and analysis in this CS domain.

Computer Science as a Domain. In recent years, interest in CS among college students in the US has dropped dramatically. However, CS and Information Technology are of enormous strategic interest, and are projected to foster vast job growth in the next few years [5]. We believe that by supporting CS education in its core we can have the largest impact on reversing the trend of students' disinterest, and on attracting women and minorities. Our belief is grounded in the observation that the rate of attrition is highest at the earliest phases of undergraduate CS curricula. This is due in part to students' difficulty with mastering basic concepts [6], which require a deep understanding of static structures and the dynamic procedures used to manipulate them [7]. These concepts

require a high level of abstraction, and the ability to move seamlessly among multiple representations, such as text, pictures, pseudo-code, and real code in a specific programming language. Thus, we believe that the availability of an ITS for basic CS would be of great benefit to both teachers and students. Such an ITS does not exist yet. This is surprising, since CS education is an area of active research, and ITSs are obviously software systems. Although ITSs on CS topics do exist, to our knowledge, only two of them tutor on the foundations. ADIS [8] tutors on basic data structures, but its emphasis is on visualization, and it appears to have been more of a proof of concept than a working system. ProPL [9] helps novices design their programs, by stressing problem solving and design skills. The other ITSs for CS focus on a diverse range of topics, from basic literacy as in AutoTutor [10], to teaching programming languages such as Lisp [11], C++ [12], and Java [13], to topics such as search algorithms used in Artificial Intelligence [14]. Of particular interest to us is the database suite of tutors composed by SQL-Tutor, NORMIT, KERMIT, and EER-Tutor [15]. These ITSs are built via constraint-based modeling, the same paradigm we chose for the development of our system.

Constraint-Based Modeling. Our system is based on a design paradigm known as *constraint-based* modeling. Originally developed from a cognitive theory of how people might learn from performance errors [16, 17], constraint-based modeling has grown into a methodology used to build full-fledged ITSs, and an alternative to the model tracing approach adopted by many ITSs. In a constraint-based system, domain knowledge is modeled with a set of *constraints*, logic units composed of a *relevance condition* and a *satisfaction condition*. A constraint is irrelevant when the relevance condition is not satisfied; it is satisfied when both relevance and satisfaction conditions are satisfied; it is violated when the relevance condition is satisfied but the satisfaction condition is not.

In the context of tutoring, constraints are matched against student solutions. Satisfied constraints correspond to knowledge that students have acquired, whereas violated constraints correspond to gaps or incorrect knowledge. An important feature is that there is no need for an explicit model of students' mistakes, as opposed to buggy rules in model tracing. Errors are implicitly specified as the possible ways in which constraints can be violated. This property simplifies the difficult and time consuming task of knowledge modeling in an ITS.

There is currently a heated debate on whether constraint-based modeling is more or less appropriate than model tracing for building ITSs [18–20]. The application of the constraint-based paradigm to a new domain can contribute to a better understanding of this issue.

Empirical Grounding. Our goal is not just to develop an ITS for CS, but to endow it with a dialogue interface that can provide more sophisticated feedback, that can help improve students' learning [21, 22]. To accomplish this goal, we are conducting an extensive tutoring dialogue collection in the data structures domain. We already collected 54 tutoring sessions, transcribed the video-recordings,

and started annotating them. More details on our data collection and preliminary analysis can be found in [23, 24]. The findings of future data analysis will guide further development of our system. The ITS we describe in this paper is in fact a baseline to which we will compare the more sophisticated and empirically grounded versions that will follow. We now describe the specific sub-domain of our research, the basic ITS we have developed so far, and its first evaluation.

2 Linked Lists

A *linked list* is a data structure used to store information sequentially. It is composed of a set of *nodes*. Each node contains two pieces of information: a *value*, representing the data we are interested in storing, and a *link* to the following node of the list. Links between nodes are realized using *pointers*, that are explicit references to the memory locations where the nodes are stored. A graphical representation of a linked list can be seen in Figure 1.

Among numerous different data structures, linked lists play a very important role in the pedagogy of basic Computer Science, making them a particularly good topic for our research. Linked lists are usually presented early in Computer Science curricula; as such, more students see this topic. According to our experience on teaching data structures in classroom, students struggle with linked lists more than with other —sometimes more complex— data structures, such as stacks and binary search trees. The fundamental concepts of linked structures, pointer manipulations, object allocation, and traversals, which students learn in the context of linked lists, are all necessary for more complicated data structures, such as trees. Linked lists are important because students can learn these concepts in a relatively simple context, and they should not cause additional cognitive overhead when students are trying to understand more complicated structures. Part of what students learn while they struggle with linked lists is to think about an abstract visual model of their data, and to think of steps in a program/algorithm as making changes to that model. Mastering that way of thinking is a huge step for students, and one that they need to make to continue successfully in Computer Science.

In the linked list domain, there are several structural properties that a solution should have in order to be correct. For example, a list should contain the correct values, as specified in the description of each problem; lists should be free of cycles; lists should not terminate with undefined or incorrect pointers; no nodes should be made unreachable from any of the variables, i.e., lost in the heap space; nodes should be correctly deleted when necessary (this applies specifically to non-garbage collected languages, like C++). Having these properties in form of constraints allows our system to catch many common mistakes students make.

3 The iList System

The iList system works by providing a student with a simulated environment where linked lists can be seen and manipulated. Lists are represented graphi-

cally, and can be manipulated with programming language commands. Students are then asked by the system to solve problems in this environment, such as insert new nodes in a given linked list, remove nodes, or perform other more complicated operations. As a student is working towards a solution, the system can provide feedback to help the student make progress.

A key difficulty about linked lists, as well as with other more sophisticated data structures, is that to really understand and use them effectively, students must think in pictures but act in code. The issue of multiple representations is subject of active research in science education [25–27]. In traditional data structures books, linked lists are illustrated with pictures, and it is sometimes difficult to connect that static representation with the dynamic procedures necessary to manipulate the structure itself. That is in fact what iList addresses. It makes the pictorial representation concrete. In a certain sense, the system reifies that conceptual image and makes it more accessible to the students. The central idea is that iList's interface is not just a box for entering input, but a dynamic visual environment that connects code actions to their effects on machine state.

Problem Types. The iList system supports two types of problems. The first kind of problems can be solved interactively, step-by-step. Students can enter a command into the system, and the system simulates the effect of that command, showing the effect of the action immediately on the simulated scenario. The second type of problems require writing a complete snippet of code, possibly involving structured conditional constructs like loops. Problems of this type usually introduce more than one initial scenario, and ask the student to write code that should work correctly in all the given scenarios. This setting forces the student to abstract away the specific details of a scenario, and think about more general algorithms for solving the problem on a wider range of situations.

The curriculum included in iList is currently composed of 7 problems, 5 of them of the first type, 2 of them of the second type. These problems have been carefully crafted based on some of the authors' experience as computer science educators, and on published CS curricula, such as ACM [7]. The goal is to challenge the students with the most common difficulties in manipulating linked lists. The problems are defined in the system using a human-readable XML format, making it easy to add new problems as needed.

Architecture. The architecture of iList is currently composed of four important modules: problem model, constraint evaluator, feedback manager, and graphical user interface. A student model and a pedagogical module, important components of a complete ITS [28], have not been implemented yet. Thus, the current version of iList is better defined as an *interactive learning environment*, rather than an ITS.

The *problem model* includes the representation of the problems presented to the student. A problem is given to the student in the form of a textual description and an initial scenario, which includes a configuration of variables and nodes (state space). The student is asked to progressively modify the state space by

interactively providing a sequence of operations, until the desired configuration of the data structure has been reached.

When the student believes he/she is done with the current problem, the current state space is submitted to the *constraint evaluator*, that checks the given solution. According to the constraint-based modeling paradigm, a solution is correct if it does not violate any constraint. Computationally, the evaluation of constraints is fairly simple. Each constraint is implemented as a computational unit with three fundamental functions: a boolean function checking the *relevance* of the constraint with respect to the solution, a boolean function checking the *satisfaction* of the constraint, and a *feedback* function responsible to return relevant information used to generate feedback for the student. A constraint is violated if the logic implication $isRelevant \Rightarrow isSatisfied$ is false for that particular state space. Constraints have access to two sources of information: the current student solution, and a correct solution provided with the problem definition. The specification of the correct solution needs only to include the minimum information necessary to evaluate a student solution, like the expected values of final lists. This is indeed one of the advantages of the constraint-based approach: the whole path towards a correct solution needs not to be specified in advance. This simplifies problem authoring, and most importantly, it allows alternative correct student solutions to be accepted by the system.

The *feedback manager* collects information from the individual constraints and builds a message directed to the student. Currently, this module simply relays messages provided by violated constraints, with minimum processing.

The *graphical user interface* is responsible for the main interaction with the student (Figure 1). The interface allows the student to interactively manipulate a data structure using C++ or Java commands. The command interpreter is quite flexible, allowing the student to focus more on the semantics of statements rather than language-dependent syntax details.

The system has been entirely implemented using the Java programming language. An early version of the system was interfaced to the WETAS system [29] for constraint evaluation. In subsequent versions, the constraint evaluator was re-implemented internally. To the user, the system appears as an applet integrated into a web page.

4 System Evaluation

A first version of the system has been deployed in a Computer Science class of a partner institution. 33 students took a pre-test before using the system, and a post-test immediately afterwards. After the post-test, the students also filled in a questionnaire about their subjective impressions on the system. The interaction of the students with the system was logged.

T-test on test scores revealed that *students did learn* during the interaction with iList (Table 1). We compared students' learning gain, defined as the difference between post-test score and pre-test score, with that of two other comparable groups of students. A group of 54 students interacted with a human

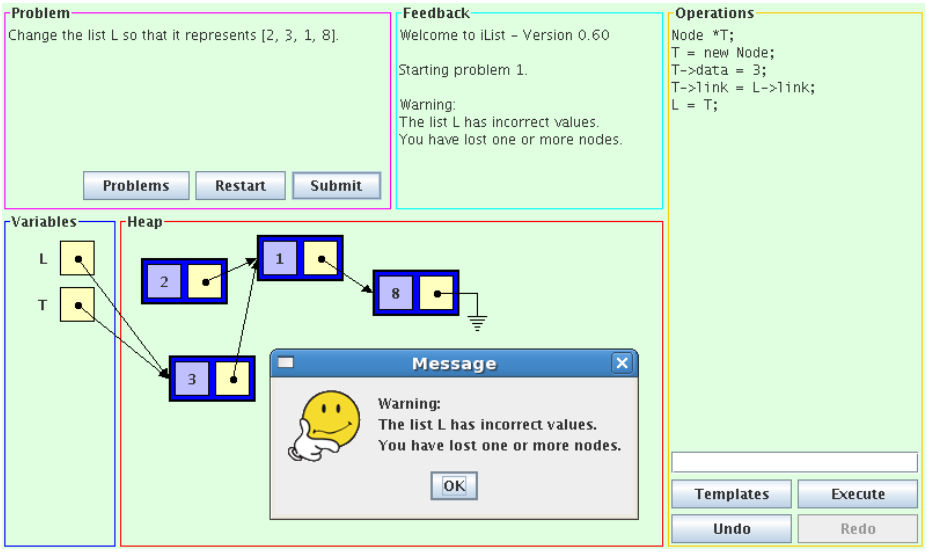

Fig. 1. A screenshot of iList

tutor between the pre and post tests. The other group (control group, 53 students) attended a lecture about a totally unrelated topic between the two tests. The tutored group achieved statistically significant learning, whereas the control group did not (Table 1).

Table 1. Test scores. Range: 0 to 1

Tutor	Pre-test score		Post-test score		Gain		T-test		
	Mean	Std dev	Mean	St dev	Mean	St dev	t	df	P
None	.34	.22	.35	.23	.01	.15	-0.56	52	ns
iList	.39	.23	.48	.27	.09	.17	-3.04	32	$< .01$
Human	.40	.26	.54	.26	.14	.25	-4.24	53	$< .01$

The learning gain of the iList group is somewhere in between the one observed in the control condition and the one of the tutored condition. ANOVA revealed overall differences between the three groups ($F(2, 137) = 5.96$, $P < 0.05$). Post hoc Tukey test indicated no significant difference between the control group and the iList group, nor between the iList group and the tutored group, whereas the difference between control and tutored groups is significant ($P < 0.01$).

The percentage of students who successfully solved each problem decreases with the problem number, as can be seen in Table 2. Problems were of increasing difficulty. Linear regression of individual problem success on learning gain showed

a positive correlation between the number of problems successfully solved and learning. Also, we found significant positive correlation between solving the most difficult problems (number 5, 6, and 7) and learning (Table 3).

Table 2. Attempt and success rates on individual problems

Problem	1	2	3	4	5	6	7
Attempt rate	100%	100%	94%	91%	77%	74%	80%
Success rate	91%	80%	74%	66%	57%	46%	31%

Table 3. Linear regression. Each line represents an independent model

Predictor	Dependent variable	R^2	β	df	F	t	P
Number of problems solved	Learning gain	.17	.41	1, 32	6.31	2.51	$< .05$
Problem 5 solved (yes/no)	Learning gain	.12	.35	1, 32	4.25	2.06	$< .05$
Problem 6 solved (yes/no)	Learning gain	.16	.40	1, 32	5.80	2.41	$< .05$
Problem 7 solved (yes/no)	Learning gain	.13	.36	1, 32	4.63	2.15	$< .05$
Questionnaire question 1	Learning gain	.22	.47	1, 31	8.33	2.89	$< .01$
Questionnaire question 4	Learning gain	.16	-.40	1, 31	5.83	-2.42	$< .05$
Questionnaire question 5	Learning gain	.37	.61	1, 31	17.72	4.21	$< .01$
Questionnaire question 6	Learning gain	.12	.36	1, 31	4.32	2.08	$< .05$
Questionnaire question 7	Learning gain	.35	-.59	1, 31	16.09	-4.01	$< .01$
Learning gain	Final class grade	.12	.36	1, 31	4.35	2.09	$< .05$

The first part of the questionnaire (Table 4) revealed that students felt that iList helped them learn linked lists to a moderate degree, and working with iList was interesting to them. The students found the feedback provided by the system somewhat repetitive, which is not surprising given the simple template-based generation mechanism. Also, the feedback was considered not very useful, but at least not too misleading.

Linear regression of questionnaire answers on learning gain revealed some significant correlations between students' feelings about the system and their learning (Table 3). The students who felt that iList helped them the most or found the feedback useful did indeed learn the most (questions 1 and 5). Those who had trouble understanding the feedback or found the feedback repetitive learned less (questions 4 and 7). Strangely, the students who found the feedback misleading learned more (question 6). A possible explanation may be that those students were more careful and exercised more critical thinking, thus getting more out of their interaction with the system.

Interestingly, students declared that they read the feedback provided by the system, but our evidence points to the opposite conclusion. From the log of the

Table 4. Questionnaire: scaled questions

Question (Scaled response: 1=No to 5=Yes)	Mean	Std dev
1. Do you feel that iList helped you learn about linked lists?	2.9	1.2
2. Do you feel that working with iList was interesting?	4.0	1.3
3. Did you read the verbal feedback the system provided?	4.3	1.0
4. Did you have any difficulty understanding the feedback?	3.0	1.5
5. Did you find the feedback useful?	2.3	1.2
6. Did you ever find the feedback misleading?	2.2	1.2
7. Did you find the feedback repetitive?	3.9	1.2

system, we estimated that students read feedback messages for 3.56 seconds on average (stdev = 2.66 seconds), resulting in a reading rate of 532 words/minute (stdev = 224 words/minute). According to Carver's taxonomy [30], such speed corresponds to the process of quickly skimming a text. According to the same taxonomy, the activity of reading to learn would require a much lower rate, in the order of 200 words/minute. Possibly, the repetitiveness of feedback messages could have made the students ignore them [31].

The last item in the questionnaire was an open response question, asking the students for any comments on the program. The detailed comments provided by the students and the instructor of the class will be helpful in guiding further improvements of the system.

Finally, linear regression revealed a positive correlation of the learning gain obtained with iList with the students' final grade in the data structure class in which they used iList (Table 3). There is then a chance that the little bit of knowledge that students acquired interacting with iList carried over to their final exam, and hopefully will help them in their future career in Computer Science.

5 Future Work

We plan to significantly extend the functionalities of iList. We will design and implement a student model, to keep track of students' history and estimate their state of knowledge exploiting the modeling power of the constraint-based knowledge representation. Pedagogical strategies will be implemented, following the results of our data analysis and those already published in the literature.

One of the research issues we are mostly interested in is the delivery of effective feedback to students. We plan to build a more sophisticated feedback module, grounding its behavior in the outcome of the analysis of our tutorial data, as well as in our past experience with the development of natural language interfaces for ITSs [21, 22]. A preliminary analysis of our human tutorial dataset suggested that *positive feedback*, i.e., reaction to correct student actions, may play an important role in tutoring [23]. We are planning on investigating the conditions and the modalities in which positive feedback is delivered by human tutors, and build a computational model of positive feedback that will be imple-

mented and evaluated in iList. Providing meaningful positive feedback in ITSs, in particular in constraint-based ITSs, is still an open problem, and a system like iList will be a useful testbed for researching that problem.

Acknowledgments. This work is supported by award N00014-07-1-0040 from the Office of Naval Research, and additionally by awards ALT-0536968 and IIS-0133123 from the National Science Foundation.

References

1. Evens, M., Michael, J.: One-on-one Tutoring by Humans and Machines. Mahwah, NJ: Lawrence Erlbaum Associates (2006)
2. Graesser, A.C., Person, N.K., Magliano, J.P.: Collaborative dialogue patterns in naturalistic one-to-one tutoring. Applied Cognitive Psychology **9** (1995) 1–28
3. Litman, D.J., Rosé, C.P., Forbes-Riley, K., Van Lehn, K., Bhembe, D., Silliman, S.: Spoken versus typed human and computer dialogue tutoring. International Journal of Artificial Intelligence in Education **16** (2006) 145–170
4. VanLehn, K., Siler, S., Murray, C.: Why do only some events cause learning during human tutoring? Cognition and Instruction **21**(3) (2003) 209–249
5. AA.VV.: US bureau of labor statistics http://www.bls.gov/oco/oco20016.htm.
6. Katz, S., Allbritton, D., Aronis, J.M., Wilson, C., Soffa, M.L.: Gender and race in predicting achievement in computer science. IEEE Technology and Society, Special Issue on Women and Minorities in Information Technology **22**(3) (2003) 20–27
7. AA.VV.: Computing Curricula 2001 – Computer Science (2001) Report of the Joint Task Force (IEEE Computer Society, Association for Computing Machinery). http://www.sigcse.org/cc2001/.
8. Warendorf, K., Tan, C.: ADIS - an animated data structure intelligent tutoring system or putting an interactive tutor on the WWW. In: Intelligent Educational Systems on the World Wide Web (Workshop Proceedings). Eighth World Conference of the AIED Society. (1997)
9. Lane, H.C., VanLehn, K.: Coached program planning: Dialogue-based support for novice program design. In: Proceedings of the Thirty-Fourth Technical Symposium on Computer Science Education (SIGCSE 03), ACM Press (2003) 148–152
10. Graesser, A.C., Person, N., Lu, Z., Jeon, M., McDaniel, B.: Learning while holding a conversation with a computer. In PytlikZillig, L., Bodvarsson, M., Brunin, R., eds.: Technology-based education: Bringing researchers and practitioners together. Information Age Publishing (2005)
11. Corbett, A.T., Anderson, J.R.: The effect of feedback control on learning to program with the Lisp tutor. In: Proceedings of the Twelfth Annual Conference of the Cognitive Science Society, Cambridge, MA (1990) 796–803
12. Kumar, A.N.: Model-based reasoning for domain modeling, explanation generation and animation in an ITS to help students learn C++. In: ITS-02 Workshop on model-based systems and qualitative reasoning for Intelligent Tutoring Systems. (2002)
13. Sykes, E., Franek, F.: An intelligent tutoring system for learning to program in Java. In: IEEE International Conference on Advanced Learning Technologies. (2003)

14. Kalayar, M., Imekatsu, H., Hirashima, T., Takeuchi, A.: An intelligent tutoring system for search algorithms. In: Proceedings of ICCE01. (2001) 1369–1376
15. Mitrović, A., Suraweera, P., Martin, B., Weerasinghe, A.: DB-suite: Experiences with three intelligent, web-based database tutors. Journal of Interactive Learning Research **15(4)** (2004) 409–432
16. Ohlsson, S.: Constraint-based student modelling. Journal of Artificial Intelligence in Education **3(4)** (1992) 429–447
17. Ohlsson, S.: Learning from performance errors. Psychological Review **103** (1996) 241–262
18. Kodaganallur, V., Weitz, R.R., Rosenthal, D.: A comparison of model-tracing and constraint-based intelligent tutoring paradigms. International Journal of Artificial Intelligence in Education **15** (2005) 117–144
19. Mitrović, A., Ohlsson, S.: A critique of Kodaganallur, Weitz and Rosenthal, "A comparison of model-tracing and constraint-based intelligent tutoring paradigms". International Journal of Artificial Intelligence in Education **16** (2006) 277–289
20. Kodaganallur, V., Weitz, R.R., Rosenthal, D.: An assessment of constraint-based tutors: A response to Mitrovic and Ohlsson's critique of "A comparison of model-tracing and constraint-based intelligent tutoring paradigms". International Journal of Artificial Intelligence in Education **16** (2006) 291–321
21. Di Eugenio, B., Fossati, D., Yu, D., Haller, S., Glass, M.: Aggregation improves learning: Experiments in natural language generation for intelligent tutoring systems. In: ACL05, Proceedings of the 42nd Meeting of the Association for Computational Linguistics, Ann Arbor, MI (2005)
22. Di Eugenio, B., Fossati, D., Yu, D., Haller, S., Glass, M.: Natural language generation for intelligent tutoring systems: A case study. In: AIED 2005, 12th International Conference on Artificial Intelligence in Education, Amsterdam, The Netherlands (2005)
23. Fossati, D.: The role of positive feedback in intelligent tutoring systems. In: ACL 2008, The 46th Annual Meeting of the Association for Computational Linguistics, Student Research Workshop, Columbus, OH (2008)
24. Ohlsson, S., Di Eugenio, B., Chow, B., Fossati, D., Lu, X., Kershaw, T.C.: Beyond the code-and-count analysis of tutoring dialogues. In: AIED07, 13th International Conference on Artificial Intelligence in Education. (2007)
25. Goldman, S.R.: Learning in complex domains: When and why do multiple representations help (commentary). Learning and Instruction **13** (2003) 239–244
26. Meltzer, D.E.: Relation between students' problem-solving performance and representational format. American Journal of Physics **73**(5) (May 2005) 463–478
27. Hundhausen, C.D., Douglas, S.A., Starko, J.T.: A meta-study of algorithm visualization effectiveness. Journal of Visual Languages and Computing **13(3)** (2002) 259–290
28. Beck, J., Stern, M., Haugsjaa, E.: Applications of AI in education. ACM crossroads (1996) http://www.acm.org/crossroads/xrds3-1/aied.html.
29. Martin, B., Mitrović, A.: WETAS: A web-based authoring system for constraint-based ITS. In Bra, P.D., Brusilovsky, P.L., Conejo, R., eds.: Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Malaga, Springer (2002) 543–546
30. Carver, R.P.: Reading Rate: A Review of Research and Theory. Academic Press, San Diego, CA (1990)
31. Heift, T.: Error-specific and individualized feedback in a web-based language tutoring system: Do they read it? ReCALL Journal **13(2)** (2001) 129–142