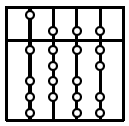
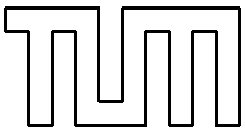


FAKULTÄT FÜR INFORMATIK
der Technischen Universität München



Lehrstuhl VIII – Rechnerstruktur/-architektur – Prof. Dr. E. Jessen

Concepts for the Implementation of Tutorial Systems in HTML and Java

Diplomarbeit

Reinhard Schaffner

Aufgabensteller:	Univ.-Prof. Dr.-Ing. Eike Jessen
Betreuer:	Dr. Michael Greiner*
Abgabedatum:	15. Mai 1998

* Neue Adresse seit 01. Januar 1998:
Siemens AG, Zentralabteilung Technik ZT PP 2S, Otto-Hahn-Ring 6, D-81730 München

Hiermit versichere ich, daß ich diese Diplomarbeit selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Reinhard Schaffner
München, den 15. Mai 1998.

Abstract

The main goal of this thesis is to examine various techniques for implementing interactive tutorial systems in the world–wide web. I commence with an overview of psychological and pedagogical issues concerned with the design of tutorial systems in general. In particular I discuss the ACT memory model proposed by J. R. Anderson and a scheme for classifying various types of educational software. I continue by examining the main building blocks of intelligent tutoring systems. In addition I discuss the use of hypertext, and cooperative working tools, the design of exercises, and user adaption.

During the growth of the world–wide web various techniques have been established to allow distance education and user interaction. This discussion will focus on methods which are suitable for implementing tutorial systems and will conclude by rating and enhancing the methods introduced. This thesis involved the development of a prototypal tutorial system using JavaScript and Cookies. A general introduction to the system is given and the design issues of the implementation and problems encountered in the development are discussed. The thesis concludes with an overview of the possible future of tutorial systems in the world–wide web.

Contents

Chapter 1

Preface

1.1 Acknowledgements

I like to express my sincerest thanks to the following people who contributed to this master's thesis (in alphabetical order). To all those who are not mentioned personally I say "thank you", because without you none of this would have been possible.

- Martin Christa, for solving technical problems and for giving me the opportunity of using his CD–ROM writer.
- Fredrik Espinoza, for being so kind as to send me his master's thesis.
- Dr Michael Greiner, for providing this fabulous LaTeX–style and supervising my work.
- Karin Hinkel, for the `xgrab` tool.
- Carol Phillips, for everything. I wish you all the best for your dissertation, and I am looking forward to receiving a copy of it.
- Kirsten Proske, for her comments and her help in testing the prototypal implementation of the Tootsie system.
- Roland Sackl, for giving me his master's thesis and for the information on how to include images in LaTeX documents.
- Thorsten Schmitt, for lending me his master's thesis and for his LaTeX introduction.
- Ed Tyson, for once reading my thesis and giving me feedback. I wish you all the best in your career.
- Christian Wenk and Irmengard Aschauer, for giving advice in regard to didactics and English grammar.

In addition, I am greatly indepted to Christian Herzog, John C. Mallery, Andrew A. Schaffner, and Gerhard Weber who kindly answered my questions concerning their research projects.

1.2 Introduction

Since the growth in personal computer use much research and testing has been carried out in the areas of computer–based training and intelligent tutoring systems, the latter often combined with, or introduced by, knowledge–based systems. As [Reinmann–Rothmeier & Mandl, 1995] state, the use of electronic media in education has been officially recommended, and so learning with multimedia has been investigated by educators and psychologists for some time. The introduction of the world–wide web has greatly increased the interest in providing distance education, fueled mainly by the enormous media attention given to the technological possibilities. Telecommunications and distant cooperation are now seen as the key to the future of education and the combining of existing tutorial systems and the internet would provide a way forward. Research into computer learning networks is still in its infancy

but net–based learning systems are advancing and offering new opportunities in learning and teaching that include multimedia but go much further.

Any lecturer wishing to publish his course on the internet must know how this can be achieved. Compared with the large number of existing training sites, the number of used techniques is small but finding the right implementation for a particular tutorial system can be very time–consuming. The questions the developer must ask himself are:

- What kind of technique will best suit his current needs?
- What are the advantages of each implementation?
- What problems may arise?
- Are the components standardised?
- What kind of exercise is he planning and will the student learn the most from it?
- Finally, how much time must be invested before training can start and how easily can the implemented system be modified or extended?

To answer all these questions and also gain basic background knowledge of psychology, pedagogy and tutorial systems in general can require a vast amount of time. This thesis is designed to assist the developer in answering these questions by providing a summary of the various techniques available and therefore more of his time can be spent on deciding upon an appropriate course structure and relevant exercises. To illustrate what can be achieved and where the restrictions lie, a prototypal implementation of a tutorial system was developed using one of the promising techniques. The system, which has been called Tootsie[1], consists of two parts, one, the developer's toolset, which is used to generate exercise files. The other is accessed by the student and displays the previously generated exercise files. The toolset is written in the programming language C, and the developer's graphical interface is developed using the common hypertext markup language, HTML, in conjunction with the scripting language, JavaScript. The tutorial system, which the student sees, also uses HTML and JavaScript. The latter provides a way to change the contents of a web page dynamically and adapt to the students needs with the help of persistent client–state information, the so–called Cookies.

The source code of the miscellaneous program, template and user interface files, which amount to more than 12000 lines of code, is not included in this master's thesis but may be obtained from the author without charge. The author disclaims all warranties with regard to that software, including all implied warranties of merchantability and fitness. In no event shall the author be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of that software. The reader is strongly encouraged to extend the current possibilities of the software, but he should keep in mind that the current version of Tootsie is a prototypal implementation whose development process was limited to a six–month period. Therefore in the following chapters different techniques will be introduced first which may be better suited to the individual developer's needs.

1.2.1 Objective of the Thesis

This thesis is mainly concerned with a comparison of different techniques used to provide user interaction and user adaption for web–based systems. Each technique is briefly introduced, and then compared with other suitable methods. Especially the benefits for educational training are pointed out in the discussion. Nevertheless the fundamental methods of each technique are specifically introduced, in order to increase the ability to understand inherent possibilities and limitations, and to acquire knowledge that is important for an implementation. The main purpose is to help the developer of a tutorial system in making a well–based and reasonable decision, while considering the advantages and disadvantages arising with a certain technique.

[1] **Tut**orial **s**ystem for **i**nteractive **e**xercises.

1.2.2 Limitations

The thesis does not discuss any technical preliminaries for making internet communication and multimedia applications possible. A general overview of techniques used in networks, such as compression algorithms and network protocols, can be found in [Heath, 1996]. The implications on society which arise from the use of modern technologies are also not covered. [Reinmann–Rothmeier & Mandl, 1995] wrote more on this subject and added a detailed reference list of publications. [Reinmann–Rothmeier & Mandl, 1995] also emphasize that humans must be taught in the use of modern technologies, if those are entering society and affecting education. Therefore it is important to know what competences are necessary then and how these can be trained. In this respect I do not intend to replace other forms of learning with the tools and techniques introduced in this thesis. Traditional textbooks, lectures, and work groups are still vital for education, and their presentation as well as their intended study goals will certainly benefit from the additional training, which tutorial systems provide.

1.2.3 Caveats

In the following points I explain how certain terms are used throughout this thesis:

- I will write "he" or "his" when speaking of a single student or system developer. This, of course, is not intended to exclude female students or system developers.
- The terms "content provider", "author", and "system developer" are used interchangeably and describe a person, who is responsible for the development of a tutorial system.
- If not stated otherwise, the terms "tutorial system", also abbreviated as "system", and "educational software" mean the same. According to the classification in Section 2.2 a "tutoring system" is a special form of tutorial system in this thesis, but not necessarily in other publications.

Chapter 2

Theoretical Background

2.1 Psychological Background

2.1.1 ACT

Anderson's ACT[1] is regarded as a general theory of cognition, with particular emphasis on skill acquisition and problem solving. In this chapter I will give a brief overview of ACT and its application in educational software, especially in the field of "tutoring", the primary purpose of which is to assist students in learning the domain of a computer–based course. As references I will use the works by [Anderson et al., 1995], [Anderson, 1996], [Wenger, 1987], [Schulmeister, 1997], and [Spada, 1992], who describe the topic in detail. In the first version of ACT, which was completed in 1982, Anderson divides the mental representation of knowledge into two categories: declarative and procedural. Declarative knowledge is organised in the form of semantic nets, which consist of smaller knowledge units, called "propositions", that describe facts, e.g. "the Earth is round", or relationships between objects, e.g. "Peter has a dog". In contrast to that, procedural knowledge, which defines the student's abilities and cognitive skills, is expressed in goal–related rules, similarily to "if–then" constructs known from programming languages. Whenever a task goal must be reached, the preconditions of the rules are tested and the goal is split up into smaller sub–goals or task states accordingly. Therefore, each rule specifies an action that must be performed or a consequence that must be considered by the student.

Acquiring declarative knowledge is not very difficult, because the propositions are directly encoded from observations or instructions. However, declarative knowledge does not enable students to solve complex problems. As mentioned before, it is the purpose of procedural knowledge to find suitable actions for the current context. Therefore, goal–independent declarative knowledge must be converted into production rules, but unfortunately this is problematic: production rules are mainly learnt by frequently applying declarative knowledge during problem solving activities (i.e. learning–by–doing). Therefore, Anderson and his research team came to the conclusion that computer–based tutorial systems which use the ACT model had to be implemented to test their hypotheses. As these sample systems are described in [Anderson et al., 1995] and [Wenger, 1987], I will not explain them here, but quote their primary intention instead:

> In human cognition, the most common form of internal modification is learning, and therefore the study of learning should be particularly revealing of cognitive structures. As environments for the investigation of learning capabilities, intelligent tutoring systems are at once flexible and predictable, two qualities which make them attractive experimentation tools ([Wenger, 1987], p290).

Long–term memory is a net of interweaved and connected propositions, which is extended by the student constantly. The size of long–term memory is in principle unlimited, but knowledge structures will be forgotten, i.e. the available information cannot be accessed anymore, if they are not continously strengthened through practice or by encoding additional and partially redundant propositions. According to

[1] **A**daptive **C**ontrol of **T**hought.

[Anderson, 1996] the activation level of a knowledge unit, i.e. its strength, controls the frequency of its use (p178). Whenever an item of information is remembered, the associated items in the same network structure will also be activated, so the stimulation is spreading along the links and their affiliated knowledge structures. This process is not only started intentionally but also subconsciously[2] (p183). If a student must elaborate the knowledge contained in a domain, challenging the student to discover and answer questions for the current subject will result in the underlying production rules and propositions being remembered better than in less active environments (pp188/193). [Schulmeister, 1997] critizises that the instructional strategies of Anderson's ACT are reminiscent of Skinner's operant conditioning and behavioristic theories[3], which are often inappropriate to teach the student complex problem solving algorithms with the help of intelligent tutoring systems (p119). However, Anderson's initial motivation in developing intelligent tutoring systems was mainly to learn more about skill acquisition than to produce practical classroom results. Nevertheless he proposed eight principles, based on his ACT theory, in order to design a computer–based instructional technology, called cognitive tutor, which supports the student in learning–by–doing. With the tutor's help skills can be displayed, monitored, and appropriate feedback can be given, in order to guide their knowledge acquisition by the system (see [Anderson et al., 1995]).

- **Principle 1: Represent student competence as a production set**
  As in behaviorism and programmed instruction a tutor separates skills into smaller components, sometimes called frames, that are presented to the learner. In contrast to the aforementioned learning methods, however, the ACT frames enable an accurate model of the target skill that allows the tutor to interpret the student's actions properly.

- **Principle 2: Communicate the goal structure underlying the problem solving**
  Although skills are split up into sub–goals or sub–tasks according to principle 1, it is often necessary to show the students explicitly what steps are required in solving a problem and how they are connected. Therefore, a tutor should use a method of reification that illustrates the structure and the relationships of goals and their sub–goals, for example, in form of a proof graph.

- **Principle 3: Provide instruction in the problem solving context**
  Instructions for the knowledge domain are best placed between each section, in which production rules are learnt by the students. Whenever a problem arises the students can return to this point, which has a fixed position between the sections. However, "[the] difficulty with this principle is that there is not a detailed theoretical interpretation of why it is true and so it is a little hard to know how to apply it in detail ([Anderson et al., 1995])". Thus, various positions for instructions must be tried: for example, students find instructions to interfer with their problem solving if these are presented at the precise point where they are needed.

- **Principle 4: Promote an abstract understanding of the problem solving knowledge**
  An algorithm for solving a particular problem is often introduced by an example. However, students tend to memorize production rules that are specifically focused on the example itself rather than the more abstract idea behind. Therefore, learners need special guidance in order to promote the creation of production rules with more general preconditions.

- **Principle 5: Minimize working memory load**
  The expression "working memory" is commonly used for information units, which are activated by mental operations in order to process a cognitive task. The capacity of working memory is normally limited to seven information units, however with the help of "chunking"[4], i.e. a mechanism that combines formerly separate units to a single "chunk", the necessary memory space can be reduced (see [Spada, 1992], p144). If a student learns a new production rule, he must keep all the relevant information simultaneously active in his memory. Consequently the size of working memory is restricting the student's ability to process all the input, therefore it is essential to minimize the load on working memory. Both, a well–designed user interface and a well–structured course, can lead to an environment, which does not interfere with human learning and understanding.

- **Principle 6: Provide immediate feedback on errors**
  This principle has provoked many discussions in the field of tutorial systems (see Section 2.1.3),

[2] In literature this is called "associative priming".
[3] See also Sections 2.2.1 and 3.3.2.
[4] Introduced by Miller in 1956.

and even Anderson's ACT theory has been modified accordingly over the last years. Formerly it was thought that production rules were formed by examining the steps which lead to a solution, but currently "[...] the learner examines the resulting solution [...] and builds productions from that. Thus, it does not matter whether all the critical steps occur together in time or not — only that they be represented in the final solution ([Anderson et al., 1995])". Therefore, immediate feedback is not necessary anymore, however it can still prevent the student from spending a long time following an erroneous path in problem solving.

- **Principle 7: Adjust the grain size of instruction with learning**
  The idea is that students combine production rules to larger units that reach the same goal by just one cognitive step. Consequently, this effects the analysis of the student's problem solving and giving instructions, because in the later stages of a course only these extended rules are observed by the tutor.

- **Principle 8: Facilitate successive approximations to the target skill**
  When students learn a new domain, they certainly do not know all the steps that lead to a solution. Therefore, it is the tutor's task to add and explain the missing parts. While the student proceeds in the course and acquires new skills, this support must be reduced by the tutor until most of the work is done by the student himself. According to [Anderson et al., 1995] this successive approximation, which results in a less and less pervasive tutor, has frequently worked quite well in practice.

2.1.2 Problem Solving

In the aforementioned section I have introduced the idea of procedural knowledge of Anderson's ACT theory. Students mainly acquire procedural knowledge during problem solving processes, because they have to divide goals into smaller sub–goals for which they know "operators". Operators are actions which can transform problem states, so a sequence of operators describes the solution to a problem. They are acquired by exploration, analogy (with an existing example), and direct instruction, for example by a human tutor. Problem solving often requires the task of searching a problem space which consists of various problem states. Therefore, a learner's goal is to find the right path through a labyrinth of states and operators. This process imposes some difficulties, because humans tend to avoid returning to previous problem states although it is necessary. In addition, they often reject operators which lead to states that at first glance differ more from the final goal than earlier problem solving stages (see [Anderson, 1996], pp235–250).

An important element of tutorial systems is the support which students receive in their problem solving process. Therefore, a tutorial system must be able to reconstruct the student's solution, in order to understand the student's reasoning. The following examples show how this problem can be resolved by creating an internal representation of the student's behaviour within a tutorial system. I will not mention the different theories of problem solving in general (like inductive reasoning, see e.g. [Spada, 1992]), but concentrate on methods which can be applied in computer–based systems to analyse the student's solution. Consequently, the main principles for modelling a student's knowledge state are:

- The system assumes from correct solutions that the various operators were correctly applied during problem solving.

- In computer–based analysis incorrect problem solving is a consequence of correctly performing an erroneous or incomplete algorithm. An alternative definition is that errors occur when students wrongly fulfil correct problem solving steps, however "[it] is more fruitful to regard the child as faithfully executing a faulty algorithm than as wrongly following a correct one (see [Spada, 1992], p207)". Errors are therefore not determined by chance, but by a rule–based, systematic, however erroneous problem solving procedure.

- The main goal is to get an individual model for each student.

The next step for a tutorial system would be to generate a step–by–step solution to any problem, whose result is then presented to the student. However, this requires the representation of cognitive structures and processes in a computer–based system. In Section 2.3.2 I will describe the intelligent tutoring system[5] Sypros, which offers this functionality, but in general the decomposition of procedural knowledge and skills depends on a detailed understanding of the knowledge domain by the system developers as well as a profound concept of implementation. The consequence is that either the domain must be limited or the generation process itself. For the simpler task of following a student's problem solving, however, [Spada, 1992] presents two methods, which are both based on the aforementioned principles:

- **Method by Brown and Burton**
  Brown and Burton represent knowledge in form of a procedural network, in which larger goals are divided into sub–goals until the granularity has reached a level, on which errors happen by chance and not by inaccuracy[6] (of the problem solving algorithm). The nodes of the network are also called procedures which are connected with "consists–of" relationships. Therefore, the algorithm of a student's problem solving can be described in detail by generating a model which solely consists of the involved procedures. A student's incorrect behaviour is reproduced by "bugs", which are represented by missing procedures, incorrect procedures, or a wrong order of procedures. However, for complex domains modelling the student's problem solving can be difficult, because for each possible bug an individual network must be generated, so due to the combinatorial complexity restrictions must be made. Another disadvantage is that the used representational language of procedural knowledge cannot explain a bug or express how the bug was acquired by the student. Its sole purpose is to replace correct and incorrect procedures (see [Spada, 1992], pp208–212, and [Wenger, 1987], pp156/157).
- **Study by Young and O'Shea**
  Young and O'Shea use in their study a system which consists of a data storage, an interpreter, and an archive of "if–then" rules, which are also called productions. In order to execute a rule the interpreter must successfully compare the elements of the data storage with the preconditions of an "if–then" rule. However, if more than one rule can be applied the interpreter will also have to resolve the conflict by deciding which rule is chosen. The student's problem solving can now be expressed in a sequence of rules, whose result is a production system that tries to match the student's solutions as precisely as possible. Missing or erroneous knowledge skills are then represented as missing rules or incorrect preconditions. In comparison to the method by Brown and Burton production systems have two advantages. First, knowledge representation is uniform, homogeneous, and modular, which means that knowledge is stored in an equal and extensible structure throughout the system. Second, modelling the differences between students requires less computation than generating networks for all the possible bugs (see [Spada, 1992], pp212–216).

The aforementioned methods, however, have a common problem: the set of correct and incorrect procedures or production rules must be known in advance before a student's solution can be diagnosed. Consequently, the system developer must include all the possible errors that might occur. For larger systems this is often not practical or even possible. However, the repair theory by Brown and Van Lehn introduces a new idea, which is based on the fact that students tend to replace missing parts of an algorithm with correct or incorrect problem solving steps creatively. The same mechanism is applied in the repair theory: the missing links are "repaired" by a problem solver with new sub–procedures, which are derived from heuristic strategies like "go back one step" etc. Rules and procedures are organised in a GAO[7] graph which [Wenger, 1987] describes as follows:

> Essentially, the basic mechanism of a GAO graph is that of a production system, with the advantages of the finely grained formalism of production rules. But this production system is interpreted with a goal stack of interspersed AND and OR goals that provide the representation with an explicit

[5] For the classification of educational software refer to Section 2.2.

[6] I.e. a systematic error which is caused for example by false training. This sentence is *not* a contradiction to the aforementioned principles of modelling students' knowledge states. It is a consequence of Brown and Burton's assumption for explaining students' errors in problem solving.

[7] **G**eneralized **A**ND/**O**R.

control structure (p166).

The rule interpreter of the production system follows the GAO graph until it reaches an impasse. In this case, the problem solver tries to repair the missing rule. The advantage is that repairs are performed locally before control is given back to the interpreter, so major reconsiderations of the algorithm are not necessary. This reduces the complexity of computation which burdens the aforementioned methods (see [Spada, 1992], pp216–218, and [Wenger, 1987], pp167/168).

2.1.3 Motivation, Feedback, and Adaption

In general, the learner's motivation can be categorised by examining their motive to approach success[8] and their motive to avoid failure[9]. According to Atkinson (see [Spada, 1992], p469) the resulting tendency of students' behaviour, **RT**, is then described by the following formula:

$$\mathrm{RT} = (\mathrm{M_S} * \mathrm{S_S} * \mathrm{P_S}) - (\mathrm{M_F} * \mathrm{S_F} * \mathrm{P_F})$$

It includes the motive disposition, which is either directed toward success $\mathbf{M_S}$ or avoiding failure $\mathbf{M_F}$, the stimulus of success $\mathbf{S_S}$ or failure $\mathbf{S_F}$, and the (subjective) probability of succeeding $\mathbf{P_S}$ or failing $\mathbf{P_F}$ a goal. Therefore, success–driven students ($\mathbf{M_S} > \mathbf{M_F}$) prefer exercises of intermediate difficulty which are oriented toward achieving a clear objective, because the product of $\mathbf{S_S}$ and $\mathbf{P_S}$ promises the highest results. On the other hand, failure–driven learners ($\mathbf{M_S} < \mathbf{M_F}$) reject these tasks in particular as the term of $\mathbf{S_F}$ multiplied with $\mathbf{P_F}$ then reaches its maximum (see [Spada, 1992], p479). The consequence is that success–driven students often set realistic goals, and whenever errors occur they reduce their expectations in order to avoid frustration. Failure–driven learners, however, choose either difficult or simple exercises, so in both cases their anticipations are confirmed: in the first case they will presumably fail, whereas in the second one the goal will certainly be reached. Here, the tutor component of a tutorial system is very important as failure–driven students tend to follow the standard which a role–model sets (despite their own experiences). In contrast to that, success–driven students reject that standard: they are either not influenced or assume, when comparing a low standard of the role–model, that their own level of expertise is higher than it really is (and vice versa). In literature, this misconception is also called contra–imitative behaviour (again see [Spada, 1992], p393).

Failure–driven students in particular depend on feedback by the tutorial system, because if it does not exist the student's level of achievement and performance will decrease (see [Harrer, 1996], p56). Early tutorial systems, which are based on Skinner's programmed instruction, apply two forms of feedback, which are given to students after they have either succeeded or failed a task. However, failure feedback which is directly presented to the learner may have been one reason why these tutorial systems were not successful. Students saw feedback less as a source of information rather than as a form of punishment. Therefore, the possibility of aversive reactions toward the system increased. In addition, motivational feedback for exercises does not have a long–term success. On the contrary, it is generally regarded as being boring, and even diminishes the students' motivation. According to [Polson & Richardson, 1988] constant intrusive feedback and advice may be decremental to instruction, and feedback that is unclear or too narrow in focus may also adversely affect learning. Instead, it should provide answers to "how am I doing?". In general, feedback must be able to find errors in a student's reasoning and report these. However, a system developer must avoid feedback being interpreted as a form of reward or punishment. Even, if feedback is not directly presented to the learners, a diagnosis of the student's current problem solving, for example with bug reports, can include a "covert" component: "covert feedback may be of more harm than instrumental use to the learner [..., it] might impede and prevent learning rather than assist it ([Schulmeister, 1997], p110)". Nevertheless, as long as feedback is not considered as a method

[8] Called "success–driven" in this thesis.
[9] Called "failure–driven" in this thesis.

of control or correction by the student, it will be accepted. Consequently, elaboration feedback is often better than verification feedback: the first is either an inherent part of the user interface, for example direct manipulation, or it gives explanations in combination with the correct answer, whereas the second one just informs the student whether his solution is right or wrong (see [Schulmeister, 1997], pp109–111).

Closely related with feedback is the ability of the tutorial system to adapt to the student, so a flexible dialog between system and learner is possible. However, implementing an "intelligent" dialog is currently difficult, because the target of adaption has neither sufficiently been researched yet nor can it be described in the formal logics of computers. Adaption is mainly concerned in presenting the information space and knowledge in a way that suits the student's individual preferences. This is either done by controlling the student's activities (also called "planned adaptivity", normally used in intelligent tutoring systems) or by giving the students the possibility to discover a broad information space on their own by giving them control over the system (also called "hermeneutical adaptivity", which is difficult to implement. The closest representative would be a hypertext system). According to [Espinoza, 1996] the issue of trust is especially important for a tutorial system. As adaption processes are just able to guess the students' intentions, he decided to choose an intermediate solution: "one way of increasing the trust in the system, is to place some control over the system in the hands of the user. [...] If the choices of the system are incorrect, the user can alter them". Partial user control is often necessary, because the granularity of the adaption process is limited. In order to reach a natural form of adaption many learner's parameters must be considered, however this could enormously increase the set of diagnosis strategies, which must be regarded by the system. Therefore, [Schulmeister, 1997] also speaks of "microadaption" (p201), because the "[...] adaptability in the best systems is rather coarse when compared to the way human teachers can weave diagnosis and didactics tightly together ([Wenger, 1987], p426)". As mentioned before, a tighter control, which results from a more detailed adaption process, hinders the students' progress, and according to [Schulmeister, 1997] contradicts learning processes. Thus, fuzzy diagnosis, which is not precise, is considered instead.

2.1.4 Didactics

Didactics is the "the art or science of teaching", which uses the following principles to choose and transform subjects of a knowledge domain into course subjects (see [Kaiser & Kaiser, 1994], pp240–262):

- Course subjects are based on current or future situations which are relevant for the students.
- Help and orientational guidance must be given to make decisions for actions in the course domain visible. Based on these, the student must then be able to manage real–life situations.
- Learning processes must be based on scientific reasoning. Therefore, the student should adopt scientific methods like the ability to accept counter–arguments or to examine a topic objectively. In addition, the possibilities as well as the limitations of scientific reasoning should be learnt.
- Course subjects are either presented as a typical example, which helps to discuss similar subjects, or in form of a specific case, which allows a general insight into a topic.
- Course subjects are organised in a reasonable course structure, often consisting of important facts, keywords, theories, models etc.

In tutorial systems these general principles are applied by the didactic module in order to adapt its presentation of topics to the needs of individual students based on the curricular information included in the course domain. The didactic module must also decide when interventions by the tutor component are necessary and what information is then given to the learner. Possible advice includes user guidance or orientational help, explanations, and tips on solving a particular problem. [Reinhardt & Schewe, 1995] recommend using the same didactic methods in a tutorial system that are already common in the domain. This makes sense because didactic methods can largely differ from domain to domain. However, they do not cover all aspects of tutorial systems as [Harrer, 1996] writes:

> Didactic models mainly give indications on which course subjects should be chosen and how long–term planning of a course is done. They provide little precise advice for helping students in their problem solving process (translated, p66).

A discussion on didactic operations for tutorial systems can be found in the book by [Wenger, 1987], pp395–415, who specialises in pedagogical activities that are intended to have a *direct* effect on the student.

2.1.5 Limitations

When evaluating tutorial systems special situations and circumstances must be regarded which can influence the results of studies that examine the effects on the student's learning process. The following points must therefore be considered:

- **Hawthorne effect**
  The Hawthorne effect describes a situation where test results are influenced by the mere fact that students are under observation. Learners often feel stimulated to increase output or accomplishments when evaluating a tutorial system.
- **Size of test groups**
  The size of a test group plays an important role, merely for statistical reasons. Results which were found after testing a few people do not have to be incorrect, but they might be irregular if applied to a different course.
- **Novelty**
  The use of the computer in learning is quite new, so most students gain their motivation for working with a tutorial system through curiosity. After a while, however, this stimulating effect must be replaced by a motivating environment that is part of the tutorial system itself. In contrast to that, the use of computers for education can also repel learners who are not familiar with this technology. Both situations must therefore be considered when evaluating tutorial systems. In addition, with the advances of computer hardware and software the results of studies, which are often not older than 10 years, are quickly out–dated. So, for example, the preference for "one window" systems, is continously decreasing as learners get more and more used to "multiple window" systems.

Consequently, results and studies in the field of educational software should be critically questioned by the reader before decisions for implementation are made. The reason is that the design process of a tutorial system requires a lot of time and energy by the system developer. For example, [Schulmeister, 1997] writes that in courseware design between 50 to 500 hours of development time are necessary for one hour of a course (p105).

2.2 Classification of Educational Software

In educational research using the computer for training has always been an important application, so over the years various software concepts were developed, which differed not only in the technologies that were available at that time, but also in their psychological and epistemological theories. As this thesis discusses the various techniques to implement a tutorial system on the world–wide web, it is first important for the system developer to know what types of tutorial systems or educational software exist. Although the following classification is based on stand–alone programs, it will nevertheless present conceptual advantages and disadvantages, which will also be valid in an network environment. Learning with the help of computer networks, however, has not been thoroughly researched yet. It still lacks the experience and the systematic research known from other learning environments. In the field of group dynamics, for example, the first observations and theoretical considerations have just been made (see [Reinmann–Rothmeier & Mandl, 1995]). The following chapter refers to the books and articles by [Weinert & Mandl, 1997], [Schulmeister, 1997], [Wenger, 1987],

[Brusilovsky & Pesin, 1996], and [Seidel, 1993], who discuss the classification of educational software in detail or introduce new ideas into the topic.

2.2.1 Drill–and–Practice and CAI Programs

Traditional drill–and–practice programs are based on the principles of programmed instruction, which received its major impetus from the work of B. F. Skinner, who described in 1954 how programs could be developed scientifically. Their main goal is to teach a knowledge domain which is well–structured, or to train students in simple skills that are suitable for mastering routine tasks. Traditionally, the student could hardly influence the linear and sequential course flow, which was defined by the system developer, so over the years other forms of training have been adopted. For these systems the ambiguous expression "computer–aided instruction[10]" is frequently used. According to [Wenger, 1987] traditional CAI programs are reminiscent books, whose contents are set in advance by the author, but which allow readers to choose the chapters they want to read individually. More sophisticated CAI systems almost resemble intelligent tutoring systems[11], as they are able to generate exercises or adapt the level of difficulty to the student's preferences and performances (pp4/5). ITS, however, uses a structural scheme[12] for the underlying system, whose components are often implemented with the help of a knowledge base. The main difference between CAI and ITS is the psychological foundation: CAI is mainly based on operant conditioning and the theory of behaviorism, while ITS is founded on cognitive psychology. In contrast to ITS, computer–aided instruction and drill–and–practice programs especially, use presentation units called frames that the tutorial system developer creates by splitting up the information space and the teacher's expertise. According to the (strictly) defined course flow these frames are displayed, and the student must normally answer a question contained in each frame. Afterwards the CAI system immediately gives positive or negative feedback, i.e. "right" or "wrong", before the next frame is shown and the same operations are repeated again. Consequently, CAI programs are better suited to domains whose information space consists of facts and whose learning goals are clearly expressed. Thus, the programs can take direct advantage of the pedagogical experience of human teachers, who must however include in the system all the possible reactions that are required in respect of the current circumstances (see [Wenger, 1987], p4). As programmed instruction is mainly used for courseware applications, further details on this theory can also be found in Section 3.3.2.

2.2.2 Tutoring Systems

As the name suggests a tutoring system is mainly aimed to support the student with an individual learning environment that acts similarly to a human tutor. Therefore, it must be flexible, dialog–oriented, and adaptable in regard to the user's input and knowledge. According to [Weinert & Mandl, 1997] the following types of tutorial systems exist:

- **Traditional tutoring system**
  Traditional tutoring systems present information about a complex subject to the student, on which questions are asked, and depending on the student's answer a new course flow is selected. Recent systems are also able to adapt to the student's knowledge and preferences. However, these implementations are rarely based on the results which were made in studies of cognitive psychology. The quality of the dialog with the student separates traditional tutorial systems from ITS, but is comparable with the knowledge presentation of sophisticated CAI programs.

- **Intelligent tutoring system**
  I will discuss the structure of intelligent tutoring systems (ITS) in the following Section 2.3, so only a short overview will be given here. In contrast to traditional tutorial systems, ITS use theories of cognitive psychology and artificial intelligence, and so they are able to give advice to the student.

[10] Abbreviated as CAI.
[11] Abbreviated as ITS.
[12] See Section 2.3.

However, applying these theories is generally difficult: it is not easy to create a suitable model for the student's cognitive structure, and it is doubtful whether the skills of a human tutor, which are not restricted to just one domain, can ever be reproduced. Nevertheless, some solutions, which are currently used in ITS research, are also found on page 18.

- **Tele–tutoring system**
  Tele–tutoring systems avoid the problems which arise in designing and programming an artificial tutor by integrating a human instructor into the system: either the student and the tutor are working on the same exercise or the tutor is contacted by the student when needed. However, as the name "tele" suggests the learner and the instructor do not have to be in the same room. In a prototypal implementation[13] the communication and the cooperation is established with the help of a computer network that transmits all the textual and audiovisual data. A disadvantage is that most of the time a tutor must be present, although the system itself can also be accessed if the instructor is absent. In my opinion, the benefits of having a human tutor on–site are certainly not restricted to tutoring systems alone. Unfortunately, this solution cannot be seen as a permanent replacement for a computer tutor. Effort must still be made in developing a good tutor model, because human instructors are not permanently available. Either the size of a group of learners or the missing resources to pay all the tutors may prevent the use of tele–tutoring systems. The technological capabilities, however, do not have to be as advanced as in the aforementioned prototypal system. For example, a similar system can be established on the world–wide web by using synchronous navigation[14] for a cooperative work environment and a chat[15] program for immediate questions and answers.

2.2.3 Hypertext and Hypermedia Systems

Hypertext–based systems belong to the category of exploratory systems. These closely resemble human thinking, because in an information domain problem solving requires the steps of searching, probing, and exploring by the students. In particular, the learner's freedom to test new solutions and the possibility to discover other areas of the domain are inevitable in hypertext–based systems. However, the success of exploratory learning is influenced by many variables: for example, it essentially depends on the student's self–confidence and competence (see [Schulmeister, 1997], p72). In psychology the theory, on which these systems are based, is called constructivism. Hereby, knowledge is dynamically created by a sub–part of the student's recognition process, which emphasizes the active interpretation[16] of an object by the learner. Therefore, knowledge cannot be simply transfered to other students without a separate reconstruction process (p74). According to [Kleinschroth, 1996] hypertext environments inherently support the acquisition of knowledge, because students can search the domain by following links, which lead to the different information units. This form of navigation is also called browsing, and it helps learners to create or reconstruct knowledge structures respectively. Which requirements must an "ideal" hypertext environment fulfil? Browsing the knowledge domain should not be restricted, so links to access all the relevant information must be offered. In addition, it should be possible to return to a familiar starting point, whenever a student is lost within the information space or reentering the system. Finally, the author must prevent learners, who are not experienced in hypertext environments or computers, having problems in using them (see [Faber, 1993]). These design guidelines are supposed to diminish a phenomenon, which is frequently called "lost–in–hyperspace". [Kleinschroth, 1996] describes two forms:

- **Museum effect**
  Essential information is "lost", missed or not found within a large information space.
- **Hansel–and–Gretel effect**
  After browsing a domain for a while users often tend to forget, what they were really looking for.

[13] MS–DOS 5.0 zum Selbststudium, developed by Siemens AG.
[14] See Section 3.2.2.
[15] See Section 4.3.2.1.
[16] In contrast to that, objectivism says that cognition, i.e. the "act or process of knowing", consists of a memory representation of objects, that is *corresponding* to the objects in the outside world.

Possible solutions to these problems are commonly aimed at the navigation support of hypertext environments, so, for example, organisational diagrams of the information space[17] or orientational indicators for the current position in the domain are used. However, [Schulmeister, 1997] asks whether the "lost–in–hyperspace" theory is a myth of pedagogical science (p59). Often authors think that a stricter form of navigation must be introduced, and they justify their hypothesis with the difficulties which learners might have in retrieving the necessary information. However, these changes also lead to a system, where hypertext links can be less freely accessed. It can be argued that a "mild disorientation can excite readers, increasing their concentration, intensity, and engagement [...]. The complete absence of orientational challenges is dull and uncomfortable. A boring hypertext is every bit as bad as a confusing one ([Schulmeister, 1997], p59)". In addition, readers of hypertext documents sometimes find information, which is very useful for them, by coincidence. However, this will be less likely if the navigation between documents is limited. The phenomenon itself is called "serendipity", the effect of which is often seen as an analogy to exploratory learning. With the aforementioned arguments it is obvious that "lost–in–hyperspace" is not an inherent part or mischief of hypertext, but a problem in the design concept of a navigation component. Disorientation and confusion can also come from the segmentation of the information space. In order to structure a knowledge domain for a hypertext environment, the various documents are split up into smaller information units, called "chunks"[18], which are then linked together to a final system by the author. However, if the size of each chunk is too small, it will be more difficult for the learner to see a coherent context between the segments. Consequently, an author must ensure that each chunk includes contextual information (see [Schulmeister, 1997], p61), like additional links to corresponding documents. According to [Faber, 1993] hypertext systems are currently seen as a more suitable basis for developing tutorial systems than drill–and–practice or CAI programs. The behavioristic learning model of CAI is often inadequate for educational software, so hypertext systems with their less hierarchical structure, which can be discovered by the students through browsing, are prefered. Although CAI has included new technologies, the importance of its underlying pedagogical concept is still neglected. The same is often said about hypertext, but it already fulfils the preconditions of exploratory learning by definiton. Partially as a consequence, adaption to the learner plays a minor role in current hypertext systems. In the paper by [Brusilovsky & Pesin, 1996], however, two techniques of "adaptive hypermedia" are introduced: adaptive presentation and adaptive navigation support. The first one distinguishes different contents for novice or expert users, so beginners get more explanations than skilled students. The second one includes an adaptive ordering technique, which arranges links in hypertext pages according to their importance. For example, the closer a link is to the top of a list, the more relevant it is for the user. Another form of adaptive navigation support is visual annotation of links as used on the pages of the UMUAI[19] journal: conferences which are held in the reader's country are specifically marked.

A paradigm for an interactive program that promotes the acquisition of cognitive structures in an exploratory learning environment is the so called "microworld". It is an artificial and closed world with its own rules, whose enclosed knowledge must be discovered by the students: "[...] a microworld may well be conceived of as a play area that gives students a chance to experiment with concepts that do not otherwise exist in a world in that combination ([Schulmeister, 1997], p51)". Based on the principles of constructivism a typical example for a microworld is Papert's Logo and Turtletalk curriculum. The students learn the programming language Logo by drawing graphs with the help of a "turtle". The turtle itself is controlled by Logo commands, and whenever it moves it leaves a trail in the form of a line on the computer screen. However, with microworlds it is not always certain whether or not the acquired knowledge can be used in real world examples, because their domain only consists of a small subset of rules. According to [Brusilovsky & Pesin, 1996] the approach of ITS and learning environments, like microworlds, is complimentary, so he suggests the combination of both: ITS will inherit the merits of an exploratory and student–driven form of learning, while microworlds, which are controlled by an intelligent tutor, could provide a more efficient system to the user. If an intelligent tutoring system is based on the microworld concept, the implementation of a tutor component will in particular be difficult, because students must be able to test their hypotheses, carry out experiments, and evaluate the results.

[17] In literature often called "maps".

[18] These chunks should not be confused with the information units of Miller's memory model (see page 5), although these hypertext chunks are aimed to reduce the work–load of the human working memory.

[19] **U**ser **M**odeling and **U**ser–**A**dapted **I**nteraction, see http://umuai.informatik.uni–essen.de/.

In this case it is better if simulations are chosen for learning ([Schulmeister, 1997], pp212/213). In their article [Brusilovsky & Pesin, 1996] describe the design of a sample application, which combines the advantages of ITS and hypertext learning environments, so I will not explain the various integration steps here. In my discussion on techniques for implementing a tutorial system on the world–wide web I will introduce the underlying hypertext model of the world–wide web, which is based on the hypertext markup language HTML (see Section 3.1). More advanced hypertext environments exist, but currently only HTML, in conjunction with a world–wide web browser[20], allows systems that are universally accessible.

2.2.4 Simulations

Computer simulations try to describe a dynamic system that exists in reality with the help of a network structure. The nodes or elements of the network represent objects of the real world. These are connected by links if the internal state of an element depends on external circumstances, i.e. other elements. The components of a network must normally be expressed in mathematical form, so a simulation is able to compute the influences on neighbouring objects, whenever a change has occurred in one element. Consequently, a lot of time and resources must be invested into the design of a network model, because otherwise all the results of a simulation may be incorrect in comparison to real events or situations. For students a simulation provides an exploratory learning environment, in which they can experiment with the different variables of the system in order to understand the underlying dependencies. This certainly promotes the creation of mental models, which [Spada, 1992] describes as large–scale knowledge units. Mental models are based on subjective observations by a person and consist of knowledge structures and processes that are used in specific situations, which are highly complex but less transparent. In problem solving mental models are very important because only if the mental representation is adequate, can a complex situation be managed by a person successfully (pp157/158). Therefore, "learning–by–doing", which is an essential part of simulations, helps students to train suitable reactions in complex environments. [Weinert & Mandl, 1997] distinguish the following goals in simulations:

- **Substitute for experiments**
  This type of simulation is often necessary if real experiments are too consumptive, expensive, or dangerous. Users can especially test their hypotheses in domains where the effects of natural processes can otherwise hardly be observed.
- **Model building systems**
  With their help users are able to write simulations by defining the underlying models themselves. Therefore, model building systems provide the tools and components, that are necessary to create an individual simulation which is then tested by the users.
- **Role playing**
  A role in an unfamiliar environment is assigned to the learner, who must try to reach a given goal by making the correct decisions. A typical example is Dörner's Lohhausen, a city in which the learner is playing the mayor's role, and has to lead the town into a prosperous future (see [Spada, 1992], p266). These simulations offer the opportunity to train in complex situations, which cannot be solved in precisely defined steps. Nevertheless learners are often highly motivated when using these systems.
- **Training of psychomotor skills**
  The simulations train psychomotor skills which must become a routine task for the students. Therefore, the system itself is very similar to reality, especially if physical information or feedback is necessary (e.g. in flight simulators).
- **Case–based learning systems**
  Case–based learning systems are very common in medical training: the students must find the correct illness and treatment for a given patient (i.e. case). Consequently, they have to analyse the information presented by the system, ask further questions, and base their diagnosis on the collected material.

[20] A program that is used for navigation in the network.

I conclude that simulations are able to offer a learning environment, which closely resembles real life situations. Their highly motivating presentation of a domain is especially an advantage compared to the other types of tutorial systems. They only lack an adaptive support for learners, but research studies must show first, how simulations can integrate a help functionality, which is able to optimize the learning process (see [Weinert & Mandl, 1997]). Nevertheless, the popularity of simulations has already led to an implementation on the world–wide web (see Section 3.6.2.2).

2.2.5 Cognitive Tools

Cognitive tools are best described as programs that help to extend or enhance human cognition. They are able to make certain tasks easier for the users, and therefore relieve human information processing from redundant work. The new capacities can then be assigned to more complex cognitive procedures, that demand problem solving skills or the student's unlimited attention. A typical example for a cognitive tool is a text processing program. With the help of WYSIWYG[21] and direct manipulation[22] the user can solely concentrate on the layout and the contents of a text, rather than worrying about printer commands etc. With these tools students are required to create knowledge structures on their own by learning cognitive concepts through exploration. Students often have to compensate for the missing structure of the underlying learning goals by planning their studies actively. In contrast to that, intelligent tutoring systems use an instructional method that starts with smaller sub–goals, which successively form a basis for higher knowledge skills. "This strongly suggests that the philosophy of intelligent tutoring is really orthogonal to the cognitive tool approach to learning ([Schulmeister, 1997], p344)".

2.2.6 Conclusion

According to [Seidel, 1993] the technical expectations in educational software are manifold: it must be portable to run on different computer platforms, support the student with an individual and adaptive environment, allow access to data which is not part of the system itself, analyse the student's actions, contain interactive components like simulations, and offer various cooperative work tools. However, [Reinmann–Rothmeier & Mandl, 1995] think that software developers often concentrate on animations or "funny graphics" rather than on didactic design and domain– or task–oriented contents. The benefits of using multimedia components in learning will decrease if the underlying pedagogical concept has not been considered in the early stages of the design process. I conclude this chapter with a subjective discussion on the various types of educational software. An overview of this discussion is presented in table 2.1 which compares the different types of educational software in regard to those aspects which are mostly required by tutorial system developers. In my opinion drill–and–practice programs, which can easily be implemented on the world–wide web, are suitable if basic skills must be taught to the students, e.g. in an introductory course. However, this does not include solving complex problems, which will be required in the learner's everyday work. For that purpose I recommend intelligent tutoring systems, preferably in conjunction with a hypertext and simulation component. On the world–wide web a hypertext system is realized with the help of HTML, which describes the layout of a document and the link structure of a course. An intelligent tutoring system can be based on HTML as well, thus incorporating the benefits of both. Building a comprehensive resource base for the hypertext component, which contains all the information the student might demand[23], is essential to enable successful learning. Simulations, which can also be embedded into a web–based tutorial system in form of Java applets, provide all the prerequisites that are needed for motivating and task–oriented training. However, the domains are mainly restricted to administrative applications, like management, or environments with simple physical rules, which rarely include natural systems because of the many unknown correlations. Cognitive tools can hardly be adapted to the world–wide web, because they require control of the user interface or the underlying browser application which is currently not possible (for example, user events for extended GUI[24] operations etc.). However, some of their concepts, especially the idea of direct

[21] **W**hat **y**ou **s**ee **i**s **w**hat **y**ou **g**et. It describes a technique, which is mainly used by text processing programs, for displaying a text on screen in exactly the same way as it will be printed on paper.

[22] A graphical user interface allows direct manipulation by representing system operations with the help of metaphors, like "drag–and–drop".

[23] See also Sections 4.2.1.2 and 4.2.3.5.

manipulation, can be used in Java applets or simulation plug–ins[25].

Table 2.1: Classification of Educational Software.

	CAI	ITS	hypertext	simulation	cognitive tool
reality[a]	training of facts, less useful outside the program.	depending on domain; close to actual tasks of learners.	depending on domain.	training of situations which are relevant to the learner.	depending on the purpose of tool.
orientation[b]	low; rarely complex and motivating problems.	depending on the implementation.	depending on the implementation.	motivating learning environment with complex problems.	depending on use of tool.
activity[c]	less active than reconstructive thinking.	active problem solving.	high; exploratory learning environment.	high; student makes complex decisions.	high; student must compensate missing learning goals.
adaptivity[d]	low; immediate feedback.	inherent part of ITS.	missing in current systems.	deficits in current systems.	low; students should have necessary skills before.

[a] Contents are based on actual requirements in the real world.
[b] Domain–specific problems that are preferably complex and motivating.
[c] Learner's role in a system.
[d] Adaptive support for the student.

2.3 General Structure of Intelligent Tutoring Systems

Based on the works by [Wenger, 1987], [Schulmeister, 1997], [Herzog, 1996], and [Gonschorek, 1997] I will shortly introduce a modular structure for intelligent tutoring systems, which is frequently used in ITS research today. For an in–depth discussion of the current topic I recommend the first two books, as I will mainly focus on a general overview that explains the basic terms. The last two authors have been working on an intelligent tutoring system called Sypros[26], which incorporates most of the methods and techniques that will be mentioned here and therefore, I will also have a closer look at the ideas behind Sypros.

2.3.1 Components of Intelligent Tutoring Systems

The main difference to systems based on computer–aided instruction is already encompassed in the term ITS itself: intelligence. Intelligent tutoring systems try to copy the skills of human tutors by focusing their concepts solely on that goal. They represent their domain knowledge in a separate part of the system, which is called the expert model as the skills and proficiency of a human expert are stored there. Knowledge is collected by applying various techniques, like interviews, questionnaires, "think aloud"–protocols, and monitoring the expert's problem solving steps. According to Anderson's ACT model two forms of knowledge are included into the expert's representation: procedural knowledge which consists of "if–then"–rules, and declarative knowledge which is best described as general facts, that are not focused on specific tasks. In addition, [Schulmeister, 1997] mentions heuristic knowledge which is based on human experts' experiences and general methods for problem solving. Knowledge of an expert model is either represented in form of a black box or glass box model. The first cannot tell the student what steps it made to solve a certain task, so its internal derivations remain invisible. The

[24] **G**raphical **U**ser **I**nterface.
[25] See page 37.
[26] **Sy**nchronisation paralleler **Pro**zesse mit **S**emaphoren, developed at Technische Universität München.

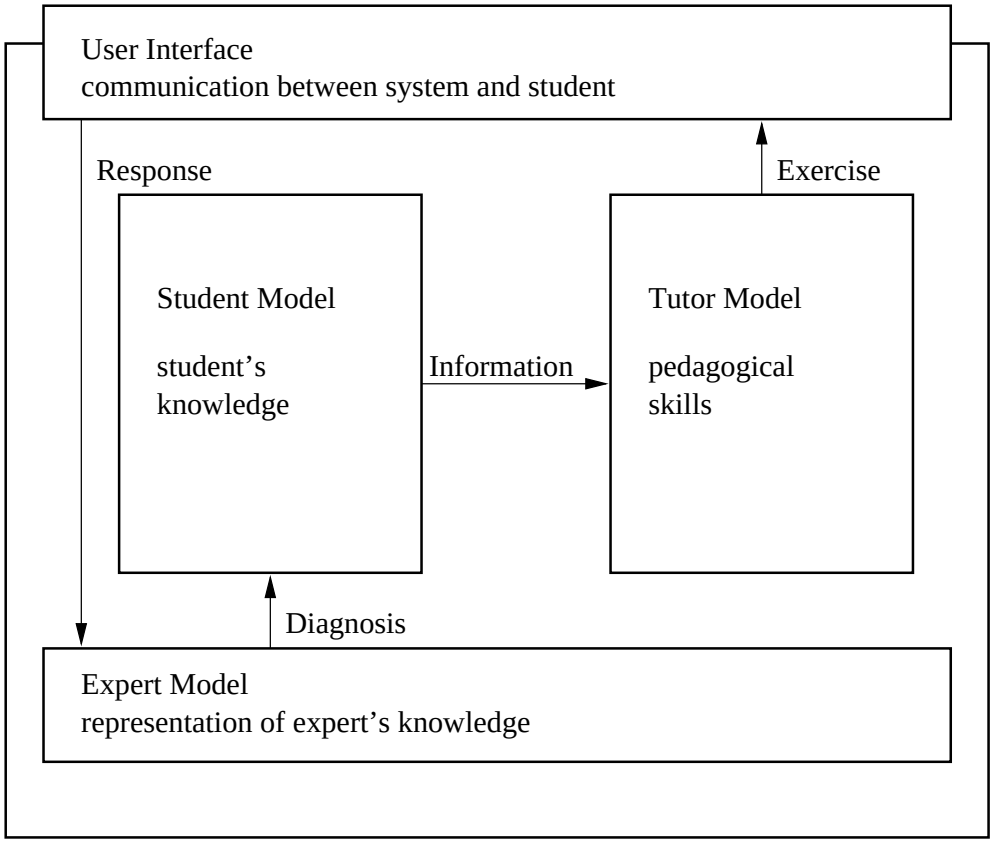


Figure 2.1: Structural scheme of an intelligent tutoring system. It consists of four different components which resemble essential parts of human education.

second one however works like a traditional knowledge–base system, and its algorithms are transparent to the student. Sometimes it is also combined with a cognitive model which is responsible for presenting all the different steps which lead to a solution in a way that closely resembles human problem solving. Unfortunately, implementing a cognitive model is even more complex than designing a glass box model, which itself is often replaced by the simpler black box model.

In traditional learning environments we do not only find an expert, but also a student, a tutor, and a method in which the student communicates with the other members. Consequently, an intelligent tutoring system contains similar concepts for each participant, thus reasonably integrating the real–life example into the world of computer–based education (see figure 2.1). The student model represents the current expertise of a student, and stores information about typical errors, problem solving methods, and the learner's preferences. The different student models in ITS research can be classified by using a scheme which is based on:

- **Information**
  A student model depends on the anticipations that it can make on the student's goals. While some systems design their student model around the final results of a task, others are able to follow the student's sub–plans that lead to a solution.

- **Representation**
  Within the system the student's knowledge is either stored as a subset or deviation[27] model. The first one keeps track of to what extend a subset of the expert's domain knowledge has been learned, while the other one monitors what errors have been made by the learner and where the student's solution differs from the expert's answer. However, both concepts use a "simplistic model of the

[27] Instead of "deviation" we also find the terms "buggy" and "perturbation" in ITS literature.

learning process" (see [SCHULMEISTER, 1997], p184), so compound bugs, i.e. mistakes that depend on each other, are hard to find, random errors are difficult to detect, and individual learning styles are rarely implemented.

- **Diagnosis**
  Diagnosing the current student's knowledge requires techniques that depend on the knowledge type, which is either procedural or declarative, and the granularity of the collected information. In ITS research model tracing, plan detection, issue tracing and "generate–and–test" exercises are frequently used. In rule–based intelligent tutoring systems model tracing tries to compute the sub–goals that lead to the student's answer. Plan detection internally creates a tree, whose root represents the problem, the inner nodes the sub–goals, and the leaves all the steps which are required to solve the problem. In contrast to that, issue tracing defines two variables for each information unit of the expert model, and their values are increased whenever the student either uses or forgets a unit. Finally, "generate–and–test" exercises are suitable to detect compound errors, because a series of exercises, which is especially designed to find a certain problem or misconception, is presented to the student.

The adaptability of an intelligent tutoring system is mainly based on the information stored in the student model, so the possibilities of the tutor model are also determined by the coverage and accuracy of the anticipated student's knowledge (see [WENGER, 1987], p16). The main task of the tutor model is to control the presentation of information. According to the differences between expert and student model on–going topics or exercises must be chosen, whose proper timing and form of presentation depend on the implemented pedagogical and didactic rules. For example, intelligent tutoring systems frequently use Socratic dialogs, in which the student is asked questions by the system in order to provoke a self–reflective analysis of errors. Coaching on the other hand is mainly concerned with introducing suitable examples and training various methods in problem solving. Therefore, the tutor model is responsible for giving advice to the student, to correct and increase the student's knowledge, and to provide motivational feedback. The epistemological concept used in current tutor models is based on instruction, and less on exploratory learning evironments, as in hypertext systems (see [SCHULMEISTER, 1997], p186). Consequently, the learner's freedom is limited, however students who are afraid of failures profit from guided learning. Nevertheless, current tutor models still have the following two disadvantages: the passive role in which the student is put by the system, and the lack of human intuition and common knowledge. [SCHULMEISTER, 1997] also criticises the fact that students are rarely asked to participate in the design process of an ITS, and although a student model assumes the student's knowledge, the learner must still develop his own expertise actively. Otherwise he will just acquire the subset of the expert model, which is given by the system, and not the proficiency of a real expert (p187).

The remaining component of an intelligent tutoring system is the user interface. Its importance should not be underestimated by the system developer, because it will be the only part of an ITS with which the student directly interacts. Firstly, the interface must present the different course topics in an understandable way, and secondly, offer the student a robust and efficient environment for learning and working with the system. An ITS is only then regarded as "intelligent" if its user interface is both flexible and adaptive to the student's needs. The system and the learner desirably communicate in a natural language[28], however it is more important that the meaning behind the student's actions is understood (also see Anderson's suggestions on page 54). [SCHULMEISTER, 1997] distinguishes four different forms of interaction:

- **Socratic dialog**
- **Coaching**
  The tutor lets the student work, and provides help only when asked.
- **Learning–by–doing**
  The tutor guides the student through the course. It suggests when to select information, and derives differences between the student and the expert model from this.

[28] Please note that: "The most natural means of communication between people is not necessarily the most 'natural' one between human and computer [...]. People are different from computers, and human–human interaction is not necessarily an appropriate model for human operation of computers ([SCHULMEISTER, 1997], p58)".

- **Learning–while–doing**
  The tutor remains in the background and occasionally gives advice.

Although in my opinion the general structure of intelligent tutoring systems is very reasonable, because the student, the expert, and the tutor are represented as in real life, some critique has been raised in regard to existing ITS implementations. [Schulmeister, 1997] quotes various research papers which say that the promises of intelligent tutoring systems are certainly overrated (pp203–220). Mainly, the primitive student model, the limitations by choosing a suitable knowledge domain, the restrictions in communication that are imposed by the current computer technology, and the hypocritical goal of tutor models to pretend that they understand the learner, are disapproved. However, these are only implementational details that may be modified or replaced in future systems. For example, an exploratory learning environment can be introduced in a tutor model without having to change the remaining parts of the system. Therefore the structural scheme itself is not concerned, but it must be clear that designing an good ITS fundamentally depends on integrating psychological and educational components which may not be available with modern computer technologies. However, it must be asked whether the abilities of human tutors are not overrated. Sometimes they also fail to give the motivational support and cognitive strategies necessary for successful problem solving.

2.3.2 Example: Sypros

The aforementioned structure has already been applied in the intelligent tutoring system Sypros, which was developed at Technische Universität München. The system is used in training students in how processes which can be executed simultaneously are synchronised with the help of semaphores. The exercises are displayed in textual form, and must be solved by defining the necessary semaphore variables and adding the suitable request and release operations. Therefore, an editor is integrated into the system, which also allows the constant monitoring of the student's actions by the system. Internally, a plan–goal–tree is built up for each exercise, in which goals represent the different methods, e.g. variable declarations or semaphore states. These lead to a plan that knows how to transform a conceptual goal into a programming construct, like semaphore operations or variable initialisations. The student's solution is continously matched with the stored plan–goal–tree, and consecutively interpreted and diagnosed according to the expert's knowledge. Furthermore, a long–term model of the student's goals is created, and it collects information on how often goals were used by the learner during the course. A visual feedback is immediately given in the form of a sad, inconclusive or smiling face. If the system is set to represent an exploratory, but guided learning environment, it will recommend further steps or explain errors when the window, in which the face is displayed, is clicked by the student. In addition, Sypros offers the possibility to simulate a synchronised program. This functionality closely resembles debuggers that are known from many programming languages: variable values are monitored, the source code is stepped through and markers can be set. Remarkably errors are automatically detected, and the simulated steps which lead to an error are shown to the learner. Also, programs are searched for conceptual errors which use too many synchronisation operations and therefore restrict allowed program runs. Finally, a hypertext–based help manual is implemented to assist the student with definitions or further explanations at any time. All these features make Sypros a useful tutoring system for training learners in its very limited domain. Nevertheless, the restriction to just one aspect in parallel programming, i.e. synchronisation, is not a disadvantage at all. On the contrary, the complexity of such a system is reduced, and therefore details, which are often demanded by critics of intelligent tutoring systems, can be implemented[29]. In my opinion, Sypros must still be seen as a tool to increase understanding of its domain rather than a replacement for a human lecturer. Hopefully one day Sypros will also be available on the world–wide web, and will certainly use the techniques that I will introduce in Chapter 3.

[29] E.g. the didactic concepts by [Harrer, 1996].

Chapter 3

Techniques

The hypertext markup language HTML is the basis for all world–wide web documents, except for programs which are solely written in Java. By definition, HTML allows the implementation of an exploratory learning environment, whose objects are realized using one of the following techniques and are embedded into an HTML document. Although the techniques are explained in separate chapters, they depend on HTML and must therefore be seen in connection with HTML to enhance the possibilities HTML already offers. Although choosing one (or more) of the following techniques is an essential part of the design process of a tutorial system, the decision for the underlying pedagogical principles and knowledge presentation is more important. The course structure and the student's progress depend on the learning environment rather than on fancy graphics, which nevertheless raise the attraction of a system if applied correctly.

3.1 Hypertext Markup Language Extensions

The invention of the terms hypertext (and hypermedia) are credited to Ted Nelson in 1965. [Hall et al., 1996] writes:

> The terms hypertext and hypermedia are often used quite interchangeably. Hypertext in the strict sense only applies to text–based systems; hypermedia is simply the extension of hypertext to include multimedia data [...]. Nelson defines hypertext as non–sequential writing and views it as a literary medium, but the ideas the term encapsulates are wider than that and include cross–referencing and the association of items (pp11/12).

In this chapter I will focus on a derivative of hypertext, the hypertext markup language, that was developed by Tim Berners–Lee at CERN in the early 1990s. It is a universally understood language for publishing documents on the world–wide web[1], where both, HTML and WWW, have largely benefited from each other: HTML is an easy–to–use language for describing the appearance of text and for distributing documents globally, and since the Mosaic browser was introduced it has blossomed with the explosive growth of the WWW. Furthermore, main features of HTML include the accessibility from anywhere and the provision of open protocols ([Hall et al., 1996]), as for each version of HTML it was tried to agree on a common standard to ensure that content providers can rely on hypertext language, which will be understood by most world–wide web browsers. The advantage for the user is that documents do not become unreadable in a short period of time ([Raggett et al., 1997]). The world–wide web is a closed hypermedia system. In contrast to open hypermedia systems[2] it combines data and hyperlink anchors in one HTML document. Unfortunately, link management is generally non–existent, and thus it is almost impossible to maintain and update information without

[1] Commonly abbreviated as WWW.
[2] E.g. Microcosm, see [Hall et al., 1996].

carefully examining the anchors[3] pointing to and from a document. Consequently, frequent world–wide web users often receive error messages, saying that the location of a document is invalid when accessing a reference in a search engine. Nevertheless Hall thinks that it is possible to implement almost any hypertext model which does not rely on embedded links in the world–wide web (p27). A sample system with better consistency and integrity of links is Hyper–G, which was developed at the Universität Graz. Links are stored in separate databases, and link management tools support the developer in keeping references updated. Unfortunately, its hypertext functionality can only be accessed through dedicated Hyper–G viewers, though client gateways to the most popular browsers by Netscape and Microsoft exist. Therefore using Hyper–G can also be considered for a tutorial system, but in this thesis I will focus the attention on techniques which will work with a broader user base. For more information on Hyper–G I like to refer to [Andrews, 1996], who covers all the basic principles.

After these introductory words I will propose what the newest version of the hypertext markup language can contribute to the design and creation of tutorial systems. In the new HTML 4.0 draft many ideas were included that will radically change accepted layout practices. These additions must be regarded as "work in progress" though, meaning that they can still be modified before the new standard is officially released. Part of HTML 4.0 are the cascading style sheets, which are discussed in an extra section of this chapter. The text continues with an outlook on HTML commands for guiding users through world–wide web documents. These tags[4] have been suggested by [Lai et al., 1995] in the article "Toward A New Educational Environment". Finally, the chapter concludes with an overview of the advantages and disadvantages of the new HTML commands in regard to tutorial systems.

3.1.1 Netscape's Layers

The competition between the two major world–wide web browser producers, Netscape and Microsoft, has frequently led to proprietary HTML commands, which were only understood by only one browser. The competitor's product just ignored them or did not display them correctly. Therefore, the use of proprietary tags is generally not recommended, because a large group of users may have difficulties with them. The new `LAYER` command by Netscape unfortunately belongs to the HTML tags, which must be rated as being problematic. With the upcoming HTML 4.0 standard it will also be replaced by a similar concept in cascading style sheets. So why do I introduce the layering technique here? The current HTML 4.0 draft is still a "work in progress", so the substitute for layers may disappear in the upcoming publications. Yet layers are very useful in designing a more intuitive layout for HTML documents. Especially in a tutorial system the presentation of information plays an important role (see Section 4.3.1).

Up to now, images were rendered in documents according to their position in the HTML code. This technique is also called "floating", as the image positions depend on the surrounding text only and cannot be fixed. Many world–wide web content designers regarded that as a problem and used the `TABLE` command or invisible images[5] as placeholders. The method is not very elegant, as the HTML code will get more complicated and difficult to maintain. Also the results often looked differently in other browsers. The layers on the other hand have fixed positions in the world–wide web page, and they can be set by the developer to overlap with remaining document texts or not. The positions can only be changed with a function written in JavaScript, whose basic concepts are described in Section 3.5.1. For example, a simple animation is programmed by setting the x–axis value of a layered image from left to right continously. Especially in a tutorial system, layers may be applied to point out ideas or to hide solutions from the student. A workable example can be found in the articles by [Burns, 1997]. Unfortunately, the following disadvantages disapprove the use of layers:

- Finding the right position for layers may require some attempts. If layers are not supposed to

[3] Strictly speaking, an anchor is the object which is the end point of a link (source or destination), and which contains the information to enable the system to locate a persistent selection within a node ([Hall et al., 1996]). In HTML the terms anchor and link are used interchangeably though.

[4] "Tag" is a synonym for "HTML command".

[5] An image which is invisible to the user with the help of transparent colours. The document text will flow around the boundaries of the image nevertheless.

overlap with the document text then extra space in the size of the image must be reserved by the developer in the document code.

- Only Netscape's Navigator 4.x is capable of displaying layers. Other world–wide web browsers will ignore the layer tag, but unfortunately not the image tags. So, all the images, which were supposed to be layered, will be shown at once, consequently often making the document unreadable.

- The layer tag will not be supported by the HTML 4.0 standardisation commitee.

The following section on cascading style sheets will show the benefits of a layering technology in layout design, but unlike Netscape's layers, moving positioned style sheets is not yet supported. Instead simple animations must be written in Java, which will require more programming skills. Luckily the W3C[6] document object model working group is currently focusing on dynamic aspects of HTML, which will include moving rendered document objects on a web page. [Furman & Isaacs, 1997] discusses that topic in detail.

3.1.2 Cascading Style Sheets

Cascading style sheets, which are abbreviated as CSS, are actually part of the new HTML 4.0 specification, but their new possibilities to define layouts for world–wide web pages require a chapter on their own. [Raggett et al., 1997] writes on the use of style sheets:

> Style sheets simplify HTML markup and largely relieve HTML of the responsibilities of presentation. They give both authors and users control over the presentation of documents [...]. Before the advent of style sheets, authors had limited control over rendering.

Style sheets can be defined by both content providers and world–wide web users. Their representation language, which has adopted characteristics of object–oriented programming like the hereinafter discussed pseudo–classes, controls the appearance of each HTML tag by assigning layout preferences, like colour or font size, to the properties of an HTML command individually. In document texts the modified HTML tag is used as in previous HTML versions, but it will now be displayed according to the settings in the cascading style sheet. Therefore, the layout and the contents of a world–wide web document can be separated from each other again. This is regarded as a breakthrough and expands the abilities of web page designers, who were getting more and more frustrated by the previous limitations of HTML. As already mentioned in Section 3.1.1 content providers tried to sidestep the stylistic limitations of HTML by using tables or images. Although the intentions to improve presentation in this way were good, some documents became unreadable for many user. [Raggett et al., 1997] remarks that style sheets:

> [...] bring back the ease of control over presentation [...]. Style sheets make it easy to specify the amount of white space between text lines, the amount lines are indented, the colors used for the text and backgrounds, the font size and style, and a host of other details.

For tutorial systems this means that the layout of exercise pages etc. can be freely designed now. For example, a style sheet, which can be loaded from an external file and shared by many HTML pages, is able to set a default to display information, e.g. text segments that appear in more than just one page, equally. By this visual support the instructor is able to focus the student's attention to the main topics or goals of a course. As style sheets can also be defined by the world–wide web users, a student will have

[6] **W**orld–**w**ide **w**eb **c**onsortium.

the opportunity to control the layout on his own, concentrating on his individual and personal needs. The rules of conflict resolution, whether the author's or student's CSS design outlines are chosen for displaying, can be found in [Lie & Bos, 1996]. I recommend to use the `important` command, which increases the weight of a style in case of a conflict, responsibly: this will deny users the opportunity of overriding the author's settings. There are two main reasons for cascading (see [Lie & Bos, 1996]):

- **Reducing redundancy**
  Style sheet definitions which have been stored in separate files can be individually combined by the world–wide web page designer.
- **Author/reader balance**
  Authors, but also readers, have equal rights to influence the presentation of world–wide web pages through style sheets. Both use the same definition language, thus continuing a tradition of the world–wide web that allows everyone to publish there.

A style rule consists of two parts: selector and declarations. In general, the selector is an HTML tag, whose property values are declared within curly brackets. If an HTML command is set between a start and end tag of another HTML command, it will inherit the styles of the surrounding element. Classes can be defined by adding a class name to those HTML tags which belong to that class. They are supposed to increase the granularity of control over elements, but: "CSS gives so much power to the `CLASS` attribute, that in many cases it doesn't even matter what HTML element the class is set on — you can make any element emulate almost any other [Lie & Bos, 1996]."

Another technique is introduced by pseudo–classes and pseudo–elements. With them, external information can influence the appearance of HTML tags. For example, the status of an HTML anchor, which is `link` (unvisited), `visited` or `active`, is regarded as being external; that means it depends on the user's previous behaviour. The difference between a pseudo–class and a pseudo–element is that pseudo–elements are part of CSS elements (like the first line of a paragraph), whereas pseudo–classes represent different types of elements (like the various states of an HTML anchor, see [Lie & Bos, 1996]). To get familiarized with the subtle distinction I will use an example from object–oriented programming: the class "car". A car consists of a motor, doors and windows. These are the sub–parts of a car and in CSS known as pseudo–elements. On the other hand, a car can be produced by more than one manufacturer. So, there will be different types of cars. In CSS words the class car would then be called pseudo–class. In our real–life example the external influence, which controls the use of pseudo–classes and elements, could be the bank account: it certainly specifies what car model we can afford. The prefix "pseudo" is used to separate these CSS elements from the previously mentioned HTML classes: pseudo–classes and elements do not exist in the HTML code, which also means that they cannot be referenced like normal classes in the document. Actually they do not have to: as I said before the choice which pseudo–class or pseudo–element is currently valid for an HTML tag solely depends on influences outside the HTML document, and not on its position in the document structure.

Each HTML element which is used within a cascading style sheet is placed into a invisible rectangular box, whose core content area and the optional surrounding padding, border, and margin areas can be defined by the developer. These format settings control the relationship to the other element boxes and the remaining document data. Such a box must still be seen as a floating text element of course, whose appearance and contents are solely specified by the developer, but not the exact location. In [Lie & Bos, 1996] many examples can be found addressing the issues of the formatting model. Another important functionality of cascading style sheets is introduced by [Furman & Isaacs, 1997] with the following words:

> Designers want to explicitly control the position of HTML elements to produce rich, static HTML documents. They also want powerful layout control to enable dynamic, animated HTML–based content. [...] relative positioning allows elements to be offset relative to their natural position in the document's flow, and [...] absolute positioning allows elements to be removed from the document's flow and positioned arbitrarily [...]. Dynamic aspects of managing positioned elements, such as

hiding, displaying and movement can only be performed using an external scripting language.

In contrast to Netscape's layers, where images are the only dynamic objects, CSS text elements can be freely positioned on a world–wide web page as well. Other positioning properties include:

- **Visibility**
  The developer can choose if a CSS text element is visible or not. In a tutorial system this functionality can be used to hide solutions or further explanations from the student in order to present these when necessary.
- **Layering order**
  It defines in which order elements are stacked upon each other. For example, it is also possible to lay text over images.
- **Overflow**
  If the contents of a positioned element exceed the reserved space, whose boundaries were declared with fixed or absolute values, the behaviour of the world–wide web browser must be specified. For example, a scrolling mechanism could be used to access the remaining data.

Many examples on behalf of positioning are found in the W3C draft by [Furman & Isaacs, 1997], but I must emphasize that the standardisation process is still in an early state. For tutorial systems positioning provides the most promising outlook. Pages are better structured and document data can be hidden or made visible adaptively. The latter techniques will require a scripting language like JavaScript though, but HTML alone is already gaining more and more influence on the user–friendly design of electronic textbooks or static HTML pages. According to [Lie & Bos, 1996] the goal has been to create a simple style sheet mechanism for HTML documents. The current specification is a balance between the simplicity needed to realize style sheets on the web, and pressure from authors for richer visual control. CSS does not offer pixel control or a layout language, which includes multiple columns with text–flow or overlapping frames. Also CSS is not expected to evolve into a programming language.

3.1.3 Hypertext Markup Language Version 4.0

After explaining the new layout possibilities, I will now have a look at the remaining features that the new HTML 4.0 standard will introduce. Here, I will focus on commands which are important for the design of tutorial systems. As there will be more modifications compared to the currently used HTML versions, I recommend the documents by [Raggett et al., 1997] be read for further information. Again I would like to emphasise that the reference is a W3C working draft for review by W3C members and other interested parties. This document may be updated, replaced or obsoleted by other documents at any time.

Until now links are generally used to visit web resources, but authors often wish to express other relationships as well. For this reason, HTML 4.0 provides new link types which help to describe the position or function of a document within a series of other documents. Among the many new types the most important are:

- `stylesheet`
  The link refers to an external style sheet. Together with the link type `alternate` which denotes substitute versions of a document the user will be able to select an alternative style sheet. For instance, the web browser may offer a pull–down menu listing all the alternatives.
- `start`
  It denotes the first document in a collection of documents. Currently tutorial system developers often use buttons in their HTML documents to achieve the same functionality. In the future, web browsers may be responsible for that.

- `next` or `previous`
  It refers to the next or previous document in a linear sequence of documents. Web browsers may choose to preload the `next` or `previous` document to reduce the perceived load time. The use in tutorial systems may be limited though, as the next pages often depend on the student's answers in the current exercise.
- `contents`, `index`, `glossary`, `appendix`, and `help`
  These are very useful link types to direct the users straight to the desired documents. For example, `index` and `glossary` lead to an index and glossary of the current document, respectively.
- `chapter`, `section`, and `subsection`
  They refer to documents serving as chapter, section or subsection in a collection of documents. In a tutorial system these may be helpful in guiding a student through the course.

There is no intention to display links that are specified by the new HTML command `LINK` and the types above, together with the contents of the remaining document text. Instead, world–wide web browser are allowed to render these in other ways, for instance as navigation tools or menu buttons. Another idea is to use the new hierarchical link types as a guide to print a series of HTML documents as a single document. In particular, certain HTML documents can be specified to serve as a table of contents or an index (see [Raggett et al., 1997]). Unfortunately the proposed link command does not support one–to–many links which are quite common in other hypertext systems like Microcosm ([Hall et al., 1996]). There, one source anchor is connected to more than one target document, which can then be selected from a menu. In my opinion there are no disadvantages against the use of one–to–many links in the world–wide web, except for more difficult link management. The missing link management in the world–wide web also makes the implementation of generic links impossible. [Hall et al., 1996] says on behalf of generic links, that they:

> [...] enable the destination of a link to be resolved at run–time calculated on the basis of the content of a source anchor rather than simply its location in a document (p7). [...] A generic link defines how to get to a document, not where to go from it (p108).

Again HTML 4.0 misses the opportunity to add generic links to the world–wide web, but implementing this concept is more complicated than one–to–many links as it will require the use of an external database or program (for example, with Hyper–G or Common Gateway Interface (CGI) programs. For an introduction to the latter see Section 3.4.1). Other HTML extensions have been made for world–wide web forms which are used to accept user input that can be processed by external programs (often Common Gateway Interface applications). These changes are mainly in the field of structuring: thematically related form controls, like checkboxes or radio buttons, can now be grouped together, so their purpose is more easily and quickly understood by users. Future plans include speech navigation of form controls to make documents more accessible for people with disabilities, however any user will benefit from this. Form controls are generally able to trigger user events, which are then processed by functions that are written by a system developer. The list of user events has been extended: they cover user actions, like changing the contents of a control item or mouse clicks, and the often needed events `mouse-move` and `key-press`. Of course, this functionality can only be used in conjunction with a scripting language like JavaScript, so further details will be mentioned in Section 3.5.1. In respect to HTML the new commands alone will definitely improve the appearance and user acceptance of the world–wide web. However, additional link types will still be needed to gain a reference structure which is more suitable for education. [Lai et al., 1995] suggests these HTML commands in his article "Toward A New Educational Environment" which will be introduced in the following section.

3.1.4 Toward A New Educational Environment

In their project [Lai et al., 1995] constructed a new educational environment, in which cooperative learning and teaching can be provided regardless of a student's location or time of access, by using the

world–wide web as their underlying medium. Despite the benefits the world–wide web provides, like accessibility and simplicity in creating web pages, they had the problem that students were often "lost–in–hyperspace". A standard world–wide web browser just allows learners to go back or forward in pages that have already been visited, but this functionality is too limited for a tutorial system. In addition, the students regularily missed documents, because they did not know of their existance, and therefore could not find them (see [Lai et al., 1995]). In order to find a solution they came up with the idea to use new HTML tags which were supposed to better organise the documents of a course. Currently an external program is responsible for giving the student an overview map and some guidance to the course by analysing the new HTML commands, but its functionality could also be included into a web browser one day. The recommended HTML tags are:

- `<PARENT HREF="...">`
  This organisation link is put in the child document to point to the parent document. Actually the idea behind this tag is very similar to the link types `chapter`, `section` and `subsection`, which are mentioned in section 24, but the implementation of HTML 4.0 will have to show that before any comparisons can be made. At the moment, the organisation link provides a simple way to tell the user, from which document the current text is derived. In this way the number of levels or ancestors is not limited.

- `<CHILD HREF="...">`
  This HTML tag is very similar to the `parent` link above. As its name suggests it will be put into the parent document to point to the child document. The tutorial system uses this information to generate a tree structure of all the documents in a course. Consequently, this can be used to point out the student's current location in a course at any time.

- `<PREREAD HREF="..." LEVEL="...">`
  Content providers can specify the HTML documents, which must be read before the current HTML page. Otherwise the student will not have the possibility to access the document. Unfortunately this tag has no equivalence in the current HTML specifications, although its advantage for tutorial systems is obvious: students will not be able to get to advanced subjects before they have read all the introductory texts. To achieve this, the web browser is responsible for storing a history of all accessed pages and the history list must not be deleted before the course is finished. In Lai's work this is currently done by an external CGI program called guider. If a student activates the guider, it will monitor the learning path of the student. For instance, if the student tries to access advanced documents, the guider will respond with a list of HTML pages that contain preliminary information. With the level argument the lecturer is able to control what documents shall be read by novices, intermediates or experts. This may help to create a tutorial system which presents users the most suitable documents in respect to their knowledge state.

Additionally Lai's guider also records the learning history of all users, so if a student has a problem he can ask who has read the documents before. Then the guider will return a list of names and email addresses. Both students, the novice and the expert, will benefit from this: novices get explanations by fellow students, while skilled students learn how to express their knowledge (also possible in a cooperative work area, see Section 4.3.2). The ideas introduced by [Lai et al., 1995] are very useful for tutorial systems. They especially help the developer to control the order in which documents are accessed by students. With web browsers currently available this must still be done externally, so the developer is required to have programming skills in a scripting language at least.

3.1.5 Discussion

This discussion is solely based on the advantages and disadvantages of HTML 4.0 and CSS for tutorial systems without any extensions like plug–ins, scripting languages or Java applets. Although these techniques rely on HTML, their benefits are mentioned in the following chapters. With the help of HTML implementing an exploratory learning environment is possible, but the hypertext system itself cannot offer any adaptive control of the students' activities.

⊕ **Integration**
In general, HTML 4.0 is the basis for world–wide web documents, so all the techniques, which are introduced in this thesis, depend on it (except for stand–alone Java programs). Therefore, plug–ins, CGI programs, JavaScript functions, and Java applets can be integrated into HTML 4.0 documents. For a tutorial system I recommend using this possibility in order to present an individual and interactive environment to the student.

⊕ **Standard**
World–wide web users and developers can rely on a single standard, which is generally accepted by the browser manufacturers, for writing and accessing world–wide web documents. HTML 4.0 even offers the opportunity to implement a common document structure on the web with the help of the new link types. For example, if a browser renders these types in the form of a pull–down menu, the user will immediately be able to select the "table of contents" of a series of documents in all the world–wide web sites which apply the new link types in their pages. In addition, HTML 4.0 is compatible to previous versions.

⊕ **Local**
HTML documents can be downloaded and stored on disk by the students. Therefore, a permanent connection to a world–wide web server is not necessary. Consequently, the work load on the server is reduced, and a slow internet connection, for example by modem, does not influence the usability of the system.

⊕ **Portability**
In general, HTML 4.0 is platform–independent, but for each operating system or computer architecture a browser which can interpret the new HTML 4.0 commands must exist. Netscape's Navigator and Microsoft's Internet Explorer partly support HTML 4.0, but their implementations differ from each other. However, this will certainly change in future releases.

⊕ **Author/reader balance**
CSS offers the possibility of overwriting the author's style sheets. Thus, students can set their own preferences for rendering documents.

⊕ **Structure**
With the help of CSS, structuring world–wide web documents becomes easier, for example a common layout can be defined for a series of documents. In addition, the separation of layout and document contents is possible.

⊖ **Availability**
At the time of writing HTML 4.0 is still a "work in progress", so it is not yet fully implemented in current world–wide web browsers.

⊖ **Interactivity**
HTML 4.0 does not offer any form of interactivity for the learner. The new event types are only available if event handlers are written in a scripting language. Consequently, if a tutorial system is solely implemented in HTML 4.0, the student and the system mostly work independently of each other.

⊖ **Link management**
The new HTML 4.0 standard does not include a better link management or one–to–many links. This functionality can only be achieved if an external program, like CGI, is used. In addition, the new HTML commands mentioned in Section 3.1.4, which provide an easy method of defining dependencies between documents, will not be standardised in the near future.

3.2 Knowledge–Based Hypertext Transfer Protocol Server

Many knowledge–based systems have been implemented during recent years, and their research is still playing an important role in computer science. Unfortunately, these systems often lack the possibility to be universally accessible, as they were mainly planned for stand–alone usage. On the other hand the

world–wide web provides global access by definiton, but it lacks the intelligent methods to store, retrieve, analyze, filter, and present information. The idea was to combine the two techniques to strengthen the links between artificial intelligence researchers and the distributed hypermedia community. In the following sections I will introduce the result of these partnerships, the Common Lisp HTTP server[7]. I will start with the programming language Common Lisp, its embedded object–oriented programming language CLOS and the presentation systems CLIM and W3P. Furthermore there will be a short overview of the hypertext transfer protocol HTTP, before the Common Lisp HTTP server is discussed. As this master's thesis is concerned with the possibility of adapting tutorial systems to the world–wide web, I will present an example which has combined an already existing tutorial system with the Common Lisp HTTP server. Finally the chapter concludes with a summary of the advantages and disadvantages of the proposed technique.

3.2.1 Common Lisp

Lisp is a programming language, which [Keith, 1997] introduces in his lecture by citing Edsger Dijkstra:

> Lisp has jokingly been called "the most intelligent way to misuse a computer". I think that description is a great compliment because it transmits the full flavor of liberation: it has assisted a number of our most gifted fellow humans in thinking previously impossible thoughts.

According to [Keith, 1997] the advantages of Lisp are: weak variable typing, a simple syntax, and programming environments resulting from 30 years of artificial intelligence research. Lisp itself is an interpreted language, which makes interactive testing possible, and it provides many facilities for symbol[8] manipulation, which is a key aspect of artificial intelligence. In addition, symbol manipulation is especially needed for building interpreters and compilers for other programming languages. Common Lisp is the official Lisp standard, and has currently gained the most influence. [Mallery, 1997] thinks that Common Lisp is one of the best available choices for developing a fine–grained vocabulary of operators (functions) that programs and software developers share. Due to a modularized structure operators do not have to be written again, and can be called by any program. This leads to high productivity because:

- Abstracted code is easily evolved as requirements change.
- Only compiling the newly added operators speeds development of large programs and facilitates evolutionary programming.
- A vocabulary of operators is build up and used in solving similar problems.

With structured and well–designed software development the same level of productivity is reached with any programming language, but in function–compositional languages like Common Lisp it is inherently encouraged ([Mallery, 1997]). Common Lisp contains a native object–oriented programming language, which is called CLOS[9]. Like many other object–oriented languages CLOS encourages the software developer to design flexible program modules for code sharing. It also supports multiple inheritance, virtual functions, and argument matching, which is necessary if functions with the same function names exist. Modern Common Lisp contains a high–level window system tool, the Common Lisp Interface Manager CLIM, which is written in Common Lisp and uses machine–independent abstractions to define window interfaces. In the final system these abstractions are replaced by specific GUI code, so the look and feel of an user interface does not change. An important CLIM abstraction is the presentation system which controls how the user perceives Lisp objects. The presentation system consists of various presentation types which specify the class methods for each object. These class methods are responsible for displaying and accepting an object or receiving user input ([Mallery, 1997]):

[7] Abbreviated as CL–HTTP server.
[8] A symbol is a physical entity, often just called entity.
[9] **C**ommon **L**isp **O**bject **S**ystem.

> In general, presentation types specialize built–in or constructed Lisp types. Presentation translators can be defined to convert from the Lisp object associated with one presentation type to another [...]. Once defined, these presentation types mediate all data entry and display, and thus, users perceive only the external, user–friendly representation, and never the internal representation.

CLIM acts as a translater between the abstract but independent user interface definitions and the system specific window commands. As this technique is not limited to traditional window systems, we can also adapt it to the world–wide web: if HTML code is returned, the user interface can be displayed in a web browser. Although CLIM is more flexible than the world–wide web user interface, it can also be configured to function in a stateless model like the world–wide web. There it must work without a persistent connection to the user, and regard the performance requirements of a server. Due to these restrictions, W3P was introduced, which Christopher Vincent describes as:

> [...] an abstract, extensible Common Lisp system for manipulating input and output as CLOS objects, allowing simpler applications with less code duplication. To facilitate compatibility with existing LISP applications, W3P implements a subset of the Lisp interface to the Common Lisp Interface Manager [...]. W3P represents an effort to create a streamlined, highly portable, non–proprietary presentation system [...].

The W3P system cannot replace CLIM, because many capabilities of CLIM do not have an equivalent in the world–wide web, but on the other hand, these missing functionalities do not become an unnecessary overhead. The W3P system uses the CLOS class inheritance, so presentation views can be defined that are specialized on different content types or styles. The same idea is used in the W3P condition and error handling: for instance, applications require different responses for input errors, so either the user is asked for the missing input data or the system proceeds without an error message, and therefore saves valuable network and server resources. Consequently, W3P is especially valuable for HTML form processing, because through its design it allows a dynamic and flexible interface that is individually generated and parsed for each user. Due to class inheritance an application can exactly specify what user input is required whenever an error has occured. Future plans for W3P include more powerful user interfaces on the world–wide web as well as client–side extensions, which allow preliminary validity checking of user input. The latter can, for example, be done with the help of JavaScript, which will be discussed in Section 3.5.1.

The proposed ideas and abilities are not solely restricted to Lisp and its derivatives. In my opinion it is possible to adapt the discussed object and presentation models for any programming language by carefully keeping the underlying design, i.e. modularity and abstraction, in mind. However, the availability of Lisp and the wide–spread use and experience in artificial intelligence research make Lisp an important option for tutorial systems.

3.2.2 Hypertext Transfer Protocol

The Hypertext Transfer Protocol, abbrievated as HTTP, was established in 1990 and is now the most wide–spread application–level protocol for distributed, collaborative, hypermedia information systems. If someone is talking about a world–wide web server, the underlying protocol is almost certainly HTTP. The first version of HTTP was a simple protocol for raw data transfer across the internet, but in the last few years the protocol has been improved. Now it allows messages to contain metainformation about the data transferred and modifiers[10] on the request and response semantics. To locate a resource in the internet HTTP uses Uniform Resource Identifiers (URI), which are often given as a location (URL) or a name (URN). The paper by [Internet Engineering Task Force, 1997] describes the overall operation of HTTP as follows:

[10] For instance, for special requirements of cache behaviour.

> The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME–like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME–like message containing server information, entity metainformation, and possible entity–body content.

The HTTP protocol is usually above the transport protocol TCP/IP, which is responsible for a connection–oriented and reliable transport of data frames. However, any other transport layer protocol can be used as long as the quality of service is similar to TCP/IP. Nevertheless, it is the responsibility of HTTP to offer a highly reliable communication channel, or failing that, a reliable indication of failure. New extensions to HTTP, that are focused on education, are suggested by [Ping–Jer et al., 1996] in the article "Synchronous Navigation Control for Distance Learning on the Web". Up to now web developers tried to restrict the pedagogical limitations of the world–wide web by using guided tours, overview maps etc., but these help tools were solely for the benefit of the student instead of addressing the whole instructor–learner relationship: "[...] they [instructors] merely construct course material, put them on the WWW, and then go behind the scenes, providing little positive aids to learners. Therefore, instructors are absent from the actual learning progress of learners." The new solution is now to provide a way in which the instructor can interact with the learner by guiding the student's navigation behaviour in the course directly. Therefore, the student gives the control of world–wide web document retrieval, browser window scrolling, mouse positioning and highlighting of texts to the lecturer, who on the other hand can decide whether he wants to advise one or many learners simultanously. The extent of navigation control must be balanced between the student and the lecturer, so the learner is involved and as a result highly motivated. Forcing the student to watch mouse movements and scrolling, thus reducing him to a passive role, is therefore not advisable. Since communication in this category involves WWW servers, HTTP is used for underlying message transmissions. The protocol specification can be found in [Ping–Jer et al., 1996]. The advantages of this technique are obvious: the tutor can show the user directly what he has to do, where the key problems are, and how he should start. This is very similar to classroom instruction, but there is no longer a need to have tutors or lecturers on–site. The developer of a tutorial system, which is based on CL–HTTP, can easily extend the CL–HTTP functionality to support navigation control as the server is programmable and the source code is freely available. Unfortunately most world–wide web browsers will not understand the navigation control requests (like the HTML extensions of Section 3.1.4).

As we can see, the work on the HTTP standard continues, however with less emphasis on educational issues than on security, caching, and connection handling. For the latter, HTTP 1.1 introduces the possibility to use a connection for more than one request–response exchange: previously a new connection was built up for each data communication and was immediately terminated afterwards. The new "server push"[11] however will keep the connection open, and so it will be interesting to see whether this new functionality will make possible the efficient use of CLIM for the world–wide web.

3.2.3 Common Lisp Hypertext Transfer Protocol Server

The Common Lisp HTTP server was specially designed by John Mallery at the M.I.T. to link artificial intelligence applications written in Lisp to the world–wide web. As knowledge–based systems often build the core of a tutorial system this HTTP server may be used as an external extension to provide tutoring and education over the internet. According to [Mallery, 1997] the main intentions to develop the CL–HTTP server were:

- **High–productivity programming**
  As a functional language Common Lisp offers advantages which have already been described in Section 3.2.1. In addition, most of the systems that John Mallery was using had been written in Lisp before, so choosing the same language for the HTTP server was a feasible decision. So, if the existing research systems grow, the possibilities of the server can therefore evolve as well.

[11] See [Netscape Developer, 1997c] for a detailed description.

- **Multiple transport media**
  John Mallery needed a technology which was able to work with different transport media, like email or HTTP.
- **Automatic form–processing**
  The research team at the M.I.T. had already gained some experience in automatic form–processing, so the same functionality was implemented in the server. Consequently, for system developers the often repetitive tasks of form–processing were reduced as well.
- **Dynamic HTML generation**
  The output of the research systems had to be processed for each user individually, so methods for flexible, dynamic, and adaptive world–wide web page generation were required and integrated.

For John Mallery programming the server in Common Lisp was the most suitable solution. Lisp makes a fine–grained vocabulary of operators possible, and can be quickly adapted to changing HTTP protocol standards. It allows rapid–prototyping and possesses datastructures that go beyond those available in scripting languages for computing and processing HTML forms. As user interaction with a tutorial system requires HTML forms for input, the latter point was essential. Additionally the appearance and the acceptance of a tutorial system depends on its user interface, so complex form processing by a fully–featured programming language is recommended. The CL–HTTP server contains all the important features of other HTTP servers or the HTTP 1.1 standard, so it is a powerful substitute and does not restrict the potential user base. It supports for example, all major HTTP methods like `GET` and `POST`, Java and JavaScript, client–side plug–ins, logging, and network security based on clients' IP addresses. Thanks to the object–oriented implementation with CLOS, generic operations can be defined for handling error conditions, or URL and server objects. In addition, W3P simplifies creating user interfaces in the world–wide web, because its presentation types describe all the data types which are either presented or received from the user.

In general, the system developer must write Common Lisp functions for computing responses to incoming HTTP requests. These functions reply to the HTTP methods `GET` or `POST` and arrange the appropriate status codes and headers, which are then returned to the client. The continuous development of the CL–HTTP server has already led to Common Lisp functions for handling Cookies[12], HTML meta information[13], and connection control, e.g. maintaining a connection to the client. Every HTTP request creates an instance of the class `SERVER` which stores all the information relevant to the transaction. Special care must then be taken with responses: as CL–HTTP can answer multiple client requests simultaneously, collisions must be prevented while accessing a shared resource.

3.2.4 Example: Episodic Learner

One application which has been based on the CL–HTTP server is the Episodic Learner Model Adaptive Remote Tutor, abbreviated as ELM–ART. It is a tutorial system which is used in an introductory Lisp course at the Universität Trier in Germany. According to [Weber & Specht, 1997] ELM–ART is based on ELM–PE, a previous application that was restricted to a small user group by its size and its platform–dependent user interface. Unfortunately it was not possible to adapt the existing system to the world–wide web directly, so with the help of the CL–HTTP server the web based version "ELM–PE" was introduced. Like its predecessor it supports example–based programming, intelligent analysis of problem solutions, and advanced testing and debugging facilities. ELM–ART has been updated since and is now available as version ELM–ART II. One of its design features is to store an individual model for each learner, which is updated automatically when a student accesses the tutorial. In the system itself pieces of information, for example texts or concepts, are structured in slots, which are called dynamic if they depend on the learner model. The other types are: static slots for prerequisites or related topics, test slots for describing a group of test items and problem slots for defining a programming problem. These information slots are used to build up units, which are then organised like a textbook into lessions, sections or terminal pages. When the student is working with the tutorial system, the individual learner

[12] Persistent client–state HTTP information, see also Section 3.5.2.
[13] Special header information, often used for "client pull" as described in the document [Netscape Developer, 1997c].

model will guide him by using "traffic lights" — the colours red, yellow and green — as a metaphor to annotate units or links in the table of contents adaptively. Therefore the units to be learned or visited next are suggested for each student according to his current state of knowledge. Additionally a technique called individual curriculum sequencing was introduced for situations when:

> [...] users may be confused about what the best next step should be to continue with the course. This may happen when the learner moves around in the hyperspace and loses orientation. Or, the learner wants to follow an optimal path through the curriculum in order to learn as fast and as completely as possible. To meet these needs, a NEXT button in the navigation bar of the text pages allows the user to ask the system for the best next step depending on the current knowledge state of the particular user ([Weber & Specht, 1997], p8).

Thanks to the CL–HTTP server ELM–ART is accessible by users all over the world, but how do the two systems work together? Gerhard Weber answered that question in an electronic mail. He wrote that the interaction between ELM–ART and the CL–HTTP server was rather simple as all the code was written in Lisp and controlled by the server. Therefore no connections to any other applications had to be established. The information stored in the individual learner model was used to create HTML pages by calling Lisp functions, so except for start or welcome pages no other HTML pages were needed in their courses. All pages that were sent to the client were solely generated within the CL–HTTP server.

3.2.5 Discussion

The main feature of the CL–HTTP server is the integration of existing knowledge bases written in Lisp into the world–wide web, so these can universally be accessed by the students.

⊕ **Availability**
As the source code of the CL–HTTP server is written in Common Lisp it can be implemented on platforms for which a Common Lisp compiler is available. This includes Windows, Apple, Unix etc.

⊕ **Integration**
According to [Mallery, 1997] the CL–HTTP server offers the support of client–side plug–ins, JavaScript, and Java. In addition, it implements the form methods `GET`, `POST` etc. which are required for the CGI program interface, but also used by Common Lisp functions to compute responses to incoming HTTP requests. Consequently, world–wide web documents can be dynamically generated.

⊕ **Extension**
The CL–HTTP extends the world–wide web providing the possibility of including knowledge–based systems which are frequently used for tutoring systems. Modifications to existing systems written in Lisp are reduced to a minimum, as the server itself uses this programming language. In addition, the server can also be extended by system developers as the source code is freely available.

⊕ **Standard**
The Common Lisp language is standarised and the CL–HTTP functions, which are included to process the students' requests and responses, are developed by the M.I.T., so the programming interface is essentially available from a single source.

⊕ **Future releases**
The CL–HTTP server is developed at the M.I.T., so it could happen that new HTTP standards or extensions may not be included in the CL–HTTP releases, but as the source code is freely available, these modifications could be made by a tutorial system developer.

⊖ **Programming experience**
Development and integration of knowledge bases requires programming experience in Lisp.

⊖ **Local**
The CL–HTTP server must collect the students' requests, query an underlying knowledge base, and compute the responses which are then returned to the clients. The work–load of a server is therefore high.

3.3 Authoring System and Courseware Plug–In

The first releases of world–wide web browsers were restricted in the data types they could display within their document window[14]. At the beginning of the internet these were regular HTML texts and `.xbm`[15] and `.gif`[16] image formats, whereas all other types of information had to be viewed with the use of external programs. Obviously this technique is not desirable for an easy–to–use and intuitive user interface: beginners had problems in setting the correct parameters for external programs, inline data and external data were displayed separately from each other, and vital system resources were necessary to call the external program, especially if the program was just required to display data. Soon the browser manufacturers came up with the idea of setting a standard to extend the abilities to present data in the browser window, without the previously mentioned disadvantages. This standard could be used by any software developer then to design lean program modules, called plug–ins, for making any type of information available to the world–wide web community without the need of external viewers. In the following sections I will introduce the general ideas behind these plug–ins, then focus on those which were especially written for computer–aided education, and conclude with an example and a summary of the advantages and disadvantages of plug–ins.

3.3.1 Plug–In Basics

I will explain the concepts of plug–ins based on the documentation and API[17] by [Netscape Developer, 1997b], especially as writing plug–ins for Microsoft's Internet Explorer is also supported by the same software package as well. Whether an API of one browser manufacturer supports all the functionalities of the competitor's counterpart may be questioned though. Nevertheless I assume that the observations mentioned in this chapter are easily adapted to a competing product by a professional plug–in developer. As explained in the introduction, a plug–in is a separate code module that behaves as though it is part of a browser. By using the API the intention is to increase the number of data types that are supported by a web browser. Firstly, this replaces external viewers because data can be displayed within the browser window now, but secondly, these extensions make a more flexible and interactive user interface possible. Therefore, it is clear that a plug–in developer needs full control of the web browser itself. All the internet functionalities such as obtaining data from the network by using URLs, must be supported as well as the receipt and handling of events triggered through user interaction. In addition, there must be methods to display data in the browser window. Custom made C/C++ functions help the developer by working with URL streams, web browser controlled memory allocation etc., so basically the programmer must decide what services he wants the plug–in to provide and which MIME[18] type and file extension it will use. However, I must emphasize here that a plug–in API is a native code library, this means that for each computer platform, on which a plug–in is supposed to run, an own proprietary API is necessary. In this respect I would again refer to the [Netscape Developer, 1997b] documentation, which contains all the signatures[19] and descriptions of the provided API functions, sample implementations, and proposed plug–in development steps. The advantages and disadvantages of the platform–specific and media type driven design are mentioned in [Netscape Developer, 1997b], but nevertheless I will summarize the most important advantages here:

- High performance because of native code implementation.

[14] Data which is rendered within the browser window is also called "inline data".
[15] X-bitmap file format.
[16] **G**raphic **i**nterchange **f**ormat.
[17] **A**pplication **P**rogramming **I**nterface.
[18] **M**ultipurpose **I**nternet **M**ail **E**xtensions.
[19] Signatures specify the name and the parameter types of functions.

- Specifically designed for extending the capabilities of a browser, so relatively simple and lightweight modules can be used.

- Plug–ins can be written in C or C++ using existing development tools.

The last but one resulted from a comparison with interapplication architectures like OLE[20] and OpenDoc. As most plug–ins support basic functionalities like displaying a special MIME type only, the lean code is understandably preferred to the overhead of the standardised OLE or OpenDoc architecture. The last argument however, which was made in connection with Java applets criticising that these require coding in a new language with new development tools, is not clear to me: the language Java is only slightly different to C/C++, and the ways of programming are very similar. Despite the benefits of platform–specific code, plug–ins inherit one severe disadvantage. In contrast to platform–native interapplication architectures and platform–independent programming languages (see [Netscape Developer, 1997b]), plug–ins cannot be simply transfered to other systems (intentionally?). Most companies, which develop plug–ins, must therefore decide whether it is feasable for them to support systems other than the wide–spread Microsoft Windows or Apple. So once more the prior intention of HTML and Java to be universally understood is threatened by the use of proprietary concepts. Unfortunately, the two plug–ins, which I will introduce in this chapter, do not run on any other system than Microsoft Windows or Apple, and no alternatives are currenly available to make courses written with the two leading toolsets for courseware production, Macromedia Director and Asymetrix ToolBook, accessible for world–wide users. Although at Technische Universität München the targeted student group for a tutorial system is mainly restricted to HP–UX or SunOS platforms, I will nevertheless discuss the possibilities of these educational plug–ins as they provide a beneficent technique for content providers to run their applications over the internet, and in addition many students have their own Microsoft Windows PC at home.

After the user has downloaded a plug–in and installed it[21], the world–wide web browser will be able to use the new module from that moment on. How will the still separate code extension work together with the browser? If the tags `EMBED` or `OBJECT` are found within an HTML page, a plug–in with a matching MIME type is looked for. In case of a successful query the module code is loaded into the memory, initialized, and a new instance of the plug–in is created. The time that a plug–in resides in memory is solely controlled by the web page, and not by the plug–in itself. As long as the surrounding web page is loaded, the plug–in will not be removed. In addition, multiple instances of the same plug–in are possible, and consequently if the last instance is deleted, the browser will free the memory space previously reserved for the plug–in code (see [Netscape Developer, 1997b]). Once a plug–in is displayed in the browser window, it does not mean that it must work independently from the other objects of the web page. In order to handle various multimedia contents and increase their capabilities, plug–ins can call Java or use Java and JavaScript controls. They can also provide parts of their functionality to other objects. The direct communication with Java and JavaScript (through a Java interface) is established with the help of LiveConnect. The highlights of LiveConnect will be explained in Section 3.5.3 later on.

Since the first introduction of plug–ins the amount of written and installed code modules has steadily grown, thanks to the ease–of–use for the web user. Other benefits include making existing applications or documents universally accessible without rewriting them, and providing interactive page contents, though these are replaced more and more by Java applets — despite their higher programming complexity (see Section 3.6.1). In addition, external programs that consume important resources are not necessarily needed anymore for simple tasks, like displaying data, when browsing the world–wide web. A good overview of available plug–ins can be found at the internet site of Netscape, whose list is updated continously.

[20] **O**bject **L**inking and **E**mbedding.

[21] This procedure has already been simplified: whenever the web browser requires a plug–in that is not currently installed, it will be made available automatically.

3.3.2 Authoring Systems and Courseware

According to [Schulmeister, 1997] authoring systems have their background in the programmed learning model, which was derived from Skinner's theories of operant conditioning (p93). Basically, the course domain is split up into tiny fractions, which are called "frames" and presented to the user. The system must wait for a reply by the student, and compare the student's answer with the right solution. In case of a correct answer the student is rewarded by the system. Authoring systems now support the developer in writing such a computer–aided tutorial by offering standarised components which describe frequently occuring structures of educational software ([Seidel, 1993]). Unfortunately, the expectations in both, the authoring toolset and the final system, were too high. Early versions just offered a linear learning path through the course, the programmed learning model was soon criticized because of its limited feedback to the learner, and the development process of applications was not simplified:

> The relatively low difficulty level in using authoring systems is a consequence of the missing programming knowledge of authors. This is possible because an authoring system imposes a strict algorithm with which the exercises are presented. Consecutively, ease–of–use is therefore paid for by limitations in performance [...]. Despite this attractive user philosophy the idea behind it did not work. [...] Because of the low demands in system and programming knowledge it is quickly overlooked that [...] authoring systems require the teacher to know at least how to split the course into tiny steps, how to combine the steps logically and consistantly, and how to give suitable feedback or reward to the learner in case of success or failure. ([Schulmeister, 1997], translated, pp103/104)

Despite the limited orientation of learning goals and lack of intuition and flexibility, [Seidel, 1993] sees two major benefits:

- Clear course objectives
- Precise methodological proceedings for students.

However, the first and very promising studies of the use of authoring systems could not withstand an in-depth evaluation, and it was even feared that poorly designed systems with lack of didactic imagination would reduce the acceptance of computer–aided instruction in general. Nevertheless, criticisms of authoring systems has led to the development of an improved version, which was called courseware. The theoretical backgrounds of authoring systems and courseware are closely related, but [Schulmeister, 1997] sees the main difference in the teaching method by quoting Jonassen: "Tutorial courseware is basically a *mis*–application of the programmed learning model of instructional design. (p107)". The strict mechanism of waiting for a student's reaction and deciding the course flow upon that was made more flexible, or it was not implemented at all. Still, interactive instruction used in courseware is often an extension of Skinner's programmed instruction, but courseware has already gained more pedagogical success than the early authoring systems ([Schulmeister, 1997], p107). Taking advantage of that, courseware is mainly concerned with the development of drill–and–practice programs, which still play a major role in computer education despite their restricted knowledge domain. Examples and design rules for courseware are mentioned in [Schulmeister, 1997], so I will not discuss these. Instead I will focus my discussion on the main commercial courseware packages by Macromedia and Asymetrix, which offer universal access to their courses by providing plug–ins for their systems.

3.3.2.1 Example: Macromedia Director, AuthorWare, and Shockwave

Macromedia Director is a wide–spread development system for any type of multimedia software. Each multimedia application is organised like a movie and contains animations, sound, video, and Lingo scripts, where all these compontents are called "members". Lingo is an object–oriented programming language responsible for user interaction and movie control that goes beyond the movie "score", which determines the role of each member. It is based on the programming language C and offers concepts like

classes, instances and inheritance. Often it is used for message handling as described by [Welsch, 1996]:

> While a movie is running various events can occur, which require a reaction by the program [...]. Director creates a message for each event that is sent to an precisely determined group of objects (i.e. scripts) for processing. Some events, which appear frequently, [...] are already defined in Lingo. The scripts, which are attached to the various objects of Director, communicate through a well designed scheme of messages, and so they allow complex reactions to different situations (translated, p267).

The main advantages of such a development system for multimedia applications, compared to standard programming languages like C++ or Java, are obvious:

- Beginners find the drag–and–drop and object–oriented user interface easier to use than having to learn a programming language first.
- As in rapid prototyping multimedia contents can be visualized quickly, and therefore development costs can be reduced by detecting design errors earlier. The scripting language Lingo misses special constructs like pointers and lacks in performance — though it reaches a remarkable 70% compared to C ([Eberl & Jacobsen, 1997], p26) — but still its functionality is appropriate for most projects.

Macromedia has introduced AuthorWare for writing tutorial systems, which should not be confused with authoring systems mentioned above. A special advantage of AuthorWare is the possibility to track the student's performance in recording the response time, the number of tries for an exercise, or the objects on which a user clicks. For example, in the variables `CorrectChoicesMatched`, `WrongChoicesMatched`, `FirstTryCorrect` and `FirstTryWrong` the information that the application needs to return individual feedback to the student is stored. After a developer has finished creating a tutorial system, AuthorWare and Director applications can be modified by the compression tool, Afterburner, to run under the Shockwave plug–in for world–wide web browsers. Generally, there are no restrictions in writing Shockwave movies except for (taken from [Eberl & Jacobsen, 1997], pp433/464):

- **Key events**
  These work within the embedded Shockwave section of the document window only if the last mouse click was in that section.
- **Lingo**
  Almost any Lingo command operates under Shockwave except for file operations that access the user's hard disk directly. This limitation was necessary to prevent security leaks via the internet, however the use of external assets[22] and preference files is still possible. Whenever these file operations are necessary, the user must explicitly confirm them, for example by downloading the external assets or answering a dialog box. In my opinion the user is left alone in making such a decision. Unfortunately there is no neutral instance rating the security issue of third–party assets or hard disk accesses, so either the user accepts an unknown application or he does not have the possibility to run it at all. A tutorial system developer must therefore keep in mind that potential users might deny doing a course, as storing the user's progress is ideally done in a preference file, which will require the user's permission.
- **Download time**
  Despite the compression tool, Afterburner, the developer of Director movies or AuthorWare tutorials must always keep the low data transmission rates in the internet or by modems in mind. Of course, this restriction will not play a major role if the targeted user group is able to access courses by a local network. However, then the advantages of remote training, e.g. running a course at any time from any place, will be lost.

[22] "Assets" are the basic materials for multimedia applications, e.g. text, image, audio or video files.

The examples at the Macromedia world–wide web site briefly show the abilities of Shockwave applications. Unfortunately at the time of writing no tutorial systems were available there except for a smaller course on how to rebuild the skeleton of a dinosaur with the use of bones displayed on screen. Bones can be picked up and dragged around. If two bones fit together then their joints are connected, otherwise the currently selected bone will fall back to its previous position. A more advanced Director application is "Word–O–Matic"[23] which introduces with circular pop up menus a new idea in the design of graphical user interfaces. As the name suggests, the commands or menu items are arranged in a circle. One positive outcome is that humans memorize for each menu item a cardinal point where they must move the mouse to reach a certain command, so after a few attempts this process is automised and menu items can be chosen even before the circular pop up menu is displayed. The use of circular pop up menus is quickly understood by the user, and as the author says: "the tools become gestures, easily learned in the user's muscle memory". As we can see here the possibilities of Director and AuthorWare do not have to be restricted in simply playing multimedia files. The usability of a web page can also benefit from the Shockwave plug–in, though a quick data transmission rate will be required.

3.3.2.2 Example: Asymetrix ToolBook II and Neuron

Working with Asymetrix ToolBook II is very similar to Macromedia Director: instead of movies the metaphor "book" is used. A ToolBook application is called "book" and consists of "pages", which are displayed in one or more windows, also known as "viewers". Like Director, ToolBook offers with OpenScript a professional programming language with commands to execute various instructions from creating new objects to establishing links to integrated Windows functions. Despite its powerful functionality OpenScript is easy to handle because of its user–friendly English–like syntax and extensive list of commands (see [HANDKE, 1997], p22). According to [ASYMETRIX, 1997] the ToolBook II authoring products include:

- **Assistant**
  With Assistant educational courseware can be created without the need of a programming language.
- **Instructor**
  Besides a drag–and–drop interface more experienced system developers can also use the scripting language OpenScript for writing courses. These can be stored as HTML documents with embedded Java applets, so students may access the system from any computer on the world–wide web.
- **Librarian**
  "[...] an Internet–based course management system that enables students to easily access courseware and administrators to track and record student progress. [...] Built on the popular Java network programming language, ToolBook II Librarian allows course instructors and administrators to monitor student activity anywhere in the world. [...] it is now possible to certify that a student has received a course; observe a student's progress; and record test results and other valuable feedback. In this way, instructors are able [to] verify each student's level of understanding enabling certification programs and accredited degrees."
- **Neuron**
  With the help of the plug–in Neuron students can access ToolBook applications on the world–wide web.

The most important feature of ToolBook II is that all its authoring products include Java applets that represent interactive questions, scoring, and feedback. Consequently, if a ToolBook application is published on the world–wide web, the Java applets will replace the proprietary ToolBook controls. These applets communicate with Librarian to capture test results and other student feedback. Therefore, a separate plug–in like Neuron is no longer necessary at the student's side. An example at the Asymetrix world–wide web site shows how a ToolBook application looks like after it has been converted into HTML and Java applets. Unfortunately its interactive and educational abilities are rather limited. However this does not have to be not be so, as another application — solely designed for Neuron — demonstrates: a

[23] See http://www.sfx.co.nz/tamahori/thought/shockers.html.

course for dentists teaches how dental x–rays are classified, or wounds are stitched. In the latter exercise, the student draws stitches on the open wound, and the accuracy of his solution is rated by the system. This example especially shows that ToolBook II suits better for writing educational training software than its competitor Director. [HANDKE, 1997] thinks that this distinction comes from the underlying concepts, because Multimedia ToolBook is organised like a book, while the Macromedia Director uses the "movie" metaphor, where various scenes must be synchronized, instead (p9). However, this could change with the AuthorWare toolset. If I compare the availability of the two authoring systems Director and ToolBook II, then the first one offers a broader user group: its Shockwave plug–in as well as the main program are available for Microsoft Windows and Apple platforms, whereas Neuron and ToolBook just support Windows systems. On the other hand, ToolBook has a major advantage in publishing applications on the internet with its included Java classes and applets, and so the missing plug–ins for other systems can be replaced by Java code. According to [EBERL & JACOBSEN, 1997] Macromedia has made a license agreement with Sun concerning the use of Java applets with Shockwave, but this is planned for future software releases (pp479/480). At the moment AuthorWare can just access JavaScript functions within an HTML page, so a connection to Java applets is already possible with Netscape's LiveConnect, but controls of an AuthorWare (as well as Shockwave) application cannot be solely replaced by applets. Unfortunately the competition between the two major companies, Macromedia and Asymetrix, has led to more software releases in a shorter period of time than ever before, so it is more difficult for developers now to adapt to all the modifications quickly and successfully ([HANDKE, 1997], p455).

3.3.3 Discussion

⊕ **Standard**
The use of plug–ins is standarized in two ways: firstly, the programming interface functions are supported by all plug–in enabled world–wide web browsers, and secondly, running an installed plug–in is solely controlled by the browser, so for the user a plug–in becomes an invisible part of the standard user interface of the browser.

⊕ **Programming experience**
In contrast to CGI and Java development, writing a courseware application is often supported by professional development tools like Macromedia's Director and Asymetrix ToolBook, which especially focus on an easy–to–use development environment for content providers who do not have programming experience. It should be noted that creating the plug–in *program* which is included into the user interface of the web browser requires advanced programming skills.

⊕ **World–wide web access**
For example, Macromedia's Authorware supports reading data files directly from the internet and displaying new HTML documents in the browser window. Consequently, the shockwave plug–in offers the same services, so a new form of user interface, like the "circular pop–up menu", could be implemented for a browser. Unlike stand–alone programs written in Java, the browser interface cannot be fully replaced, so I recommend extending the possibilities of the interface only if necessary. Most world–wide web users are already familiar with the browser interface, so changing this standard could hinder the learning process of students.

⊕ **Local**
As a plug–in must be installed on the user's computer (and not on a world–wide web server) it is inenvitably exectued locally. The same is valid for plug–in applications which do not have to download files from the internet, but contain all the relevant data. Consequently, the work–load of a world–wide web server is reduced compared with CGI programs.

⊕ **Extension**
Plug–ins extend the possibilites of world–wide web browsers to display new file formats or run applications written with external programs.

⊕ **Interactivity**
As the examples in the previous sections show, plug–ins offer real interaction between the student and the tutorial system.

⊖ **Availability**
Plug–ins are native code extensions, so they are specifically written for certain operating systems and computer architectures. In addition, a plug–in enabled world–wide web browser is necessary. For example, Macromedia's Shockwave plug–in only runs on Windows 95 and Apple PowerPC.

⊖ **Portability**
The native code implementation of plug–ins prevents them being portable between different computer systems. However, a plug–in application, like a Shockwave movie, is portable between installations of the same plug–in.

⊖ **Download size**
If complex applications and user interfaces are realised with the help of a plug–in application, the size of the file which must be downloaded quickly increases. As the usability of a tutorial system also depends on the time the students have to wait for a reaction from the system, the sizes of plug–in applications should be limited depending on the speed of the internet connection.

⊖ **Security**
Macromedia's Authorware offers two security levels for its applications: "trusting" and "non–trusting". For example, in the "non–trusting" mode of the Shockwave plug–in writing to the user's hard disk and downloading external commands or libraries is prohibited[24]. Whenever the user accesses an application which requires the services of the "trusted" mode, a dialog box asks for confirmation to execute the program. However, this decision is solely left to the user as there is no authority which can objectively tell whether an application can be trusted or not. The problem is that tutorial systems which often store a permanent student model require an "agreement of trust" by the student.

3.4 Common Gateway Interface

"The combination of adaptivity and hypermedia on the World Wide Web is in technical terms a tricky one. [...] Few possibilities exist in present tools [...] providing the flexibility that is needed when tailoring information to individual users ([Espinoza, 1996])". One of these is the Common Gateway Interface[25], which was introduced with the first HTTP servers to provide access to external programs from the world–wide web. Since then it has expanded into a powerful method to perform user requests, and only with the programming language Java an equivalent alternative has been established. Using the Common Gateway Interface is amazingly simple as long as a tutorial system developer has some experience in programming.

3.4.1 Common Gateway Interface Basics

Common Gateway Interface programs can be written by a system developer in any programming or scripting language, as long as the result is executable and running independently under a world–wide web server. Therefore, we call CGI programs "external applications" as they are not part of an HTTP server itself. Consequently, they do not depend on a special server, thus making them interchangable between different HTTP server implementations. With the help of CGI applications it is also possible to access programs that do not have their own world–wide web interface. In this case the CGI application must work as a gateway between the two different environments: user input and program output are processed to modify the data according to the required formats. For example, HTML code can be added to the program output, so the result is then rendered by a world–wide web browser. Common Gateway Interface programs are often needed when the contents of an HTML form must be processed, because a CGI program can be executed in real–time and output dynamic information[26]. Basically, the communication between the user and the form processing application works like this: the world–wide web user fills in the form with individual data and submits it to a server. The server receives the data

[24] An exception: in "non–trusting" mode external world–wide web documents which are not on the user's hard disk can be read. Writing is still strictly forbidden.
[25] Abbreviated as CGI
[26] I.e. according to the user's input data is generated on–the–fly. As with normal HTML files the contents of an output cannot be changed after these have been displayed in the web browser window.

and knows by the information stored in the URL what external program must be started or queried, if the CGI application is running continously. The output of the program is collected by the server then and sent back to the user's world–wide web client. If no errors have occurred the user will see the response to his request in the browser window.

After these introductory words I will have a deeper look at the technology behind the Common Gateway Interface. CGI data is accepted by the world–wide web server with the help of two methods, `GET` and `POST`, which are chosen by the system developer depending on the purposes of each HTML form. The difference between the two methods is small, but important: the method `GET` is intended for transmitting small amounts of data to the server, as the data is appended to the URL that symbolises an external program call. This technique has one major benefit though. Due to the appended data each external program call is distinct, so the user is able to set a bookmark or a link to the requested resource without ever having to re–enter the data again. Therefore, this method is commonly applied in internet search engines in order to return a link to the user, which then can be processed by the search program each time the user selects that link, as all the required parameters are already set. The method `POST` does not offer this functionality as data is separated from the URL, however this makes transmissions of unrestricted length possible. The data is sent to the standard input stream of the external program, and parsed there. In the latter case especially, environment variables, which are additionally set by the world–wide web server for the current data stream, are needed to send parameters to the program. Amongst the most important are ([December Communciatons, 1997]):

- `CONTENT_LENGTH`
  Primarily used by the method `POST`, it denotes the length of the content as given by the client.
- `CONTENT_TYPE`
  Primarily used by the method `POST`, it represents the content type of the data for queries that have attached information. For example, the type `application/x-www-form-urlencoded` for form data.
- `QUERY_STRING`
  Used by the method `GET`, it contains the information following the first `?` in the URL which references the external program. The symbol `?` separates the URL from the data, and the query string is encoded in the standard URL format by changing spaces to `+` and encoding special characters with `%xx` hexadecimal values. Therefore, the string must be decoded before it can be processed.
- `REMOTE_ADDR`
  The IP[27] address of the remote host making the request.
- `REQUEST_METHOD`
  It stores the method by which the request was made, e.g. `GET` or `POST`.

The system developer should check these variables before analysing the data with the CGI program. Further precautions are especially necessary if the source code of the program contains disk operations or handles sensitive data. Normally, a CGI application is executed under the user identity of its owner, so all the owner's rights are also available for the CGI program. The developer must therefore prevent any misuses, which could alter the proper operations of a program. While the program is running, data can be sent to the standard output stream continously. This output can either be an HTML or text document generated by the program, or instructions to the server for retrieving a desired output. It is important that at least some output must be given as otherwise an error message will be displayed on the user's screen. The output begins with a small header which contains the so called server directives. Headers which are not server directives are directly sent back to the client. The most common directive is `Content-type`, which denotes what MIME type the returned document has. For example, in case of HTML, the developer must put `Content-type: text/html` in the header. The final header is then separated from the remaining data by a blank line.

The following examples will show how Common Gateway Interface programs are already used in distant education and educational training. They are a powerful and easy–to–use method for software developers,

[27] **I**nternet **P**rotocol, a connection–less protocol for data transmission.

in order to combine new or already existing tutorial system with the world–wide web. At the moment CGI applications are the only world–wide web extensions, which can be accessed by any browser available, as long as plain standardised HTML code is returned. The potential user group is unrestricted, and a system developer will largely profit from the fact that just one version of the tutorial system must be implemented (new technologies are often limited to the browsers by Microsoft or Netscape, so other browsers would require special solutions or alternate versions). However, there are some disadvantages:

- **Programming knowledge required**
  As previously mentioned, the Common Gateway Interface applications must be written in a programming language like C/C++ or Perl. However, many potential tutorial system developers may not have programming experience at all, so in this case courseware authoring systems are recommended.
- **High work–load on servers**
  Whenever the client sends a request to the world–wide web server, an instance of the CGI program is started to compute the response. If many users are accessing the system at the same time, then each new instance will increase the work–load of the server, and so increase the response time as well. For a training course, whose access rate statistics contain certain peak points repeatedly, the resulting delays may influence the usability and user acceptance of the whole system. However, the work–load can be reduced by preventing incorrect user input for example. With the use of JavaScript the contents of form entry fields can be checked, before data is sent to the CGI program. If JavaScript is not available then the necessary data checks must be done by the CGI program itself, hereby consuming important system resources.
- **Limited interaction**
  Interaction with Common Gateway Interface programs is limited to request–response communications, so the prospects of a tutorial system will also depend on the response time of the network. In addition, it is not possible to implement applications which require direct user manipulation like the dinosaur construction kit mentioned on page 37. As long as interaction can be limited to data that is selected with the help of checkboxes or radio buttons, or entered by the student as text, Common Gateway Interface applications can be used. If a tutorial system requires mouse events or constant student monitoring, then Java, JavaScript (partially) or plug–ins are recommended. The CGI program must output the code for embedding Java applets, JavaScript functions or plug–in objects, but it will no longer be compatible to browsers which do not implement these methods.

The Common Gateway Interface method alone does not provide all the abilities required by successful tutorial systems. However, it forms a reliable basis for an adaptive and intelligent tutorial system, onto which an interactive user interface can be built with the help of Java or partially JavaScript. The CGI program is responsible then for managing domain and expert knowledge, initiating the tutoring component, and storing user models, whose information is collected by queries to the CGI program as well as by reports coming from Java applets and JavaScript functions. In addition, already existing tutorial systems can be transformed into CGI applications, primarily by changing the way in which the user input and system output are handled. For example, if an intelligent tutoring system was structured with the architecture suggested in Section 2.3, then only the communication module would require new programming, in order to use world–wide web displaying and interaction techniques.

3.4.2 Common Gateway Interface in Education

The uses of Common Gateway Interface programs are manifold, and the following examples are just representatives for all the applications that exist in educational training.

3.4.2.1 Example: Virtual Seminar Koalah

The seminar Koalah[28] last held at the Ludwig–Maximilians–Universität München in the winter semester 1996/97[29] is completely computer–based and managed over the world–wide web, so students from

[28] **Koo**peratives **A**rbeiten und **L**ernen **a**n der **H**ochschule.

different universities can participate. At the beginning of the seminar the members are assigned to a group, in which they work on a common task or topic, without having to meet personally. In fact, each group gets its own discussion board, where the members of that group can join to make proposals, publish results or contact the lecturer and other seminar members. Whenever a student wants to make a contribution to the board, he must do that with the help of a special email link. In that link the suggested subject line of the email is given, so the posted messages are structured by default: for example, by the subject line alone the type of a message, e.g. question or answer, can be identified by the server and handled accordingly. The messages mailed to the world–wide web server are in fact not parsed by a Common Gateway Interface program, but a similar technique is used. Various other examples of virtual seminars, in which discussion boards are implemented by CGI applications, exist but the herein introduced seminar Koalah has already been the focus of a research project, whose experiences were summed up in a paper by [Nistor & Mandl, 1995]. The results will be presented hereinafter. The seminar itself is structured into different project steps, like introduction, project analysis, and final discussion. Each exercise uses various mile–stones to synchronise the group members with the overall schedule. Discussions are commented and rated by the lecturer, who alone decides when a mile–stone was reached. In addition, a chat room[30] was implemented to provide a way for informal meetings.

What were the experiences of the virtual seminar? [Nistor & Mandl, 1995] say that on the technical side the world–wide web, in which most of the information was retrieved, was so slow that an important part of the seminar, which was searching for data, was difficult to handle. After the students had familiarised themselves with the navigation in the world–wide web, selecting the right information was also a problem. Therefore, the lecturer was mainly occupied with organising the seminar rather than focusing on topical aspects of the discussions. The text–based exchange of information was actually hindering the communication rather than promoting it. The seminar members were mainly concerned with the form and contents of their messages, and the students felt that information was primarily reflected rather than exchanged. During the seminar students continually expressed their demand for more social interaction, as they had no contact with the other seminar members. Exercises were not seen as a common goal but as an individual task, and students did not feel the need to solve them in time. In addition, the communication within a seminar group was mainly centered around the lecturer. [Nistor & Mandl, 1995] came to the conclusion that more social interaction and an early introduction to cooperation were important to improve the communication amongst seminar members. With the online chat room and the mile–stones they have already done that, though in my opinion a better structured discussion board should also be adopted. By the use of CGI programs the student could select in which order he would like to have the subject lines of messages displayed, or whether the discussion board is structured hierarchically[31] or not.

3.4.2.2 Example: LaTeX–Tutorial

The LaTeX–tutorial[32] is a representative of all the Common Gateway Interface scripts or applications that work as an interpreter between an external program and the world–wide web. Other examples are the CGI programs used by internet search engines that transform a user's query into an appropriate database query and send back HTML code for displaying the results. The LaTeX–tutorial is primarily intended for beginners, who want to make their first steps in this document preparation system which uses complex commands to describe text layouts. The student learns with the help of trial–and–error and immediate feedback, how command modifications influence documents. First the new commands are presented to the student in an introductory text, where examples for closer inspection can be selected. Whenever this happens, a world–wide web form is loaded into the browser which contains the LaTeX code of the selected example in a text entry box. The code itself can be edited by the student, so he is able to change the parameters, or add and remove commands. If the button for submitting the form is clicked, the LaTeX code is sent to a CGI program that transmits the data to the LaTeX compiler. It collects the textual output of the compiler, and, if the compilation was successful, it initiates the conversion of

[29] See http://infix.emp.paed.uni–muenchen.de/nic/ws9697/tnseite9697.html.
[30] A chat room is a virtual meeting point where people can talk online in a primarily text based environment.
[31] I.e. replies immediately follow after the originating message.
[32] See http://www.uni–giessen.de/hrz/tex/cookbook/zero.html.

the result — normally a `.dvi` file — into a graphics format that can be displayed by the world–wide web browser. Afterwards an appropriate HTML frame is generated to present both files to the student. There he can see what influences his modifications had. In spite of the simple solution the LaTeX–tutorial is a remarkable example on how CGI programs can be used in education. The system is individually tested and examined by the student as it allows modifications of the sample code and feedback by the LaTeX compiler. However, one improvement could be made: in case of an error the textual output of the compiler should be analysed by the CGI program to support the student with a better explanation than the standard error message does.

3.4.2.3 Example: Plan and User Sensitive Help

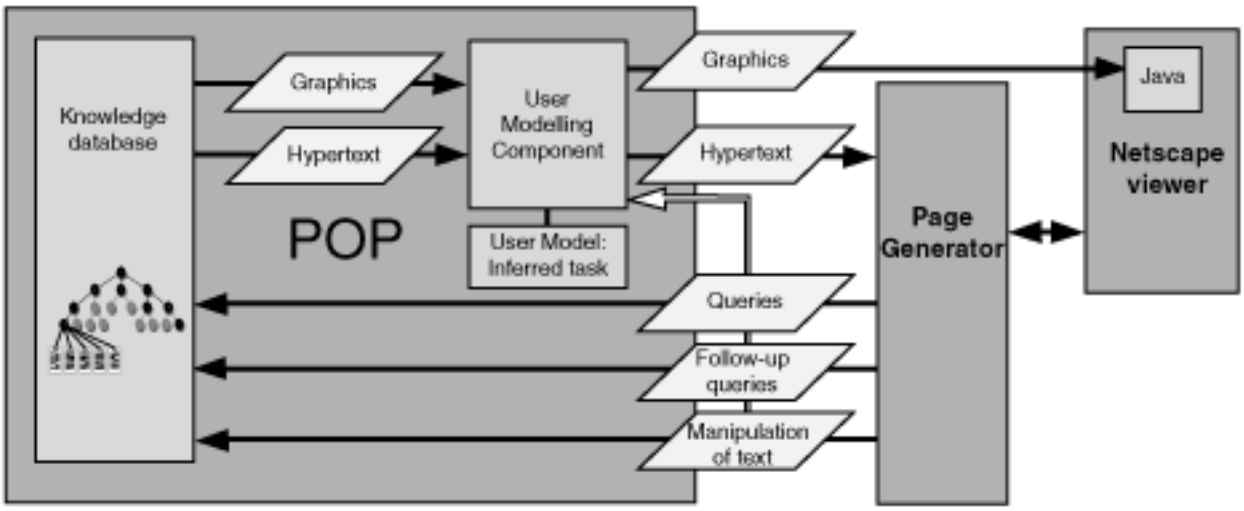


Figure 3.1: Architecture of the Plan and User Sensitive Help system. The browser, called "Netscape viewer", sends requests to the page generator which retrieves data from the knowledge base and adapts the presentation of an hypertext page according to the information received from the user modelling component. Graphics, which show the user's current position in the information space, are directly sent to a Java applet in the browser window.

The PUSH[33] project is a test system, which provides an adaptive user interface for searching and retrieving manuals on the software development method SDP[34]. Its main goal is to reduce the risk of an information overflow, that may happen due to the enormous amount of documents which are stored in the underlying database. According to [Höök, 1996] information is presented in a structure that is closely based on the domain itself. Follow–up questions and hotwords establish links to alternate documents that are specifically oriented to the current user's task. Therefore, PUSH tries to determine what intentions a user has, and individually decides, what information is presented. I will split the introduction to PUSH, which was developed at SICS[35], into two parts: one, that discusses the use of Common Gateway Interface programs here, and another one in Section 3.6.2.1, that will focus on the PUSH Java applet. In figure 3.1 the overall architecture is shown. POP[36] is an adaptive hypermedia system with a world–wide web based interface to the information in the SDP manuals, and according to [Espinoza, 1996] it has been implemented with the SICStus Prolog programming environment. The Prolog database is queried by a page generator, that retrieves the information which the user currently needs. In the result HTML code is included, and so a document whose contents are adaptively selected by the generator is presented to the reader. The data is structured with the help of HTML,

[33] **P**lan and **U**ser **S**ensitive **H**elp.
[34] **S**ystem **D**evelopment **P**rocess.
[35] SICS is a non–profit research foundation funded by the Swedish National Board for Technical and Industrial Development (NUTEK) and by a group of companies (CelsiusTech AB, FMV, Ellemtel Utvecklings AB, IBM Svenska AB, Sun Labs, Ericsson and Telia AB).
[36] **P**USH **O**perational **P**rototype.

and various link and control elements are added to enhance the text with clickable buttons, menus and follow–up references (see [Espinoza, 1996]). The page generator is a CGI program, which dynamically creates world–wide web pages. The advantage of this method is that individually generated pages are returned to the user without having to keep a static database of all possible queries and their resulting HTML files on disk. In addition, the document data, which is stored in the database, and the code for the interactive user interface are separate from each other (see [Espinoza & Höök, 1996]), so new SDP documents can be added or changed without the need to modify the page generator itself. Another design constraint was that information, which was returned, had to be restricted, because world–wide web users tend to read the part of a document, which is shown in the browser window at once, i.e. without the use of scroll bars (see [Espinoza, 1996]). Therefore, unnecessary data is not immediately displayed to the user, but marked as "collapsed stretch–texts" in the HTML page. The user can click on these markers, and the hidden information is expanded. Due to the design of the generated pages a new query to the slow Prolog database is not required anymore: in the first query all the available information is included, but commented out by the page generator. In any consecutive queries the comments are removed, and the reformatted page is directly sent back to the user's web browser. More details on how the page generator and the Prolog process interact with the use of sockets, and the web browser and the generator with CGI, can be found in the thesis by [Espinoza, 1996].

A similar method, i.e. using Common Gateway Interface programs to scan for markers in files, is often applied when implementing proprietary extensions to current world–wide web standards. The article by [Lai et al., 1995], which was introduced in Section 3.1.4, describes a prototypal implementation of a web browser, which processes the proposed HTML tags with the help of CGI programs. Whenever a student accesses an HTML page of the tutorial system, the contents of that HTML file are scanned for the new tags. According to the specification either a hierarchical overview structure is generated (for the commands `PARENT` and `CHILD`), or a list of preliminary pages which have to be read before access to the currently selected HTML file is granted (for `PREREAD`) is displayed.

3.4.3 Discussion

⊕ **Availability**
Of all the techniques introduced in this chapter CGI programs have the best availability: as long as they only return HTML code to the student, any world–wide web browser can render the results. In contrast to that, plug–ins, JavaScript and Java just work with Netscape's Navigator and Microsoft's Internet Explorer.

⊕ **Portability**
As the output of CGI programs can be displayed by any world–wide web browser, the students can access CGI applications from any computer platform for which a browser exists. The CGI application, however, is generally not portable because it is an external program which is compiled and runs on a specific platform. Consequently, the portability depends on the source language, in which a CGI application was written.

⊕ **Standard**
The CGI interface is fully standardised within HTTP, because the output of a CGI program must consist of an HTTP header and the document text.

⊕ **Integration**
As CGI programs normally return HTML documents, plug–ins, JavaScript code, Java applets etc. can be embedded into the document source. In addition, the CGI interface allows integration of existing applications into the world–wide web: the system developer must then provide an intermediate interpreter which modifies the data streams from and to the external application accordingly.

⊕ **Extension**
CGI mainly extends the possibilities of the world–wide web by making external programs available. This is particularly important for tutorial systems: an intelligent tutoring system can be realised in which the communication between system and student is established by a CGI program while

the remaining parts of the system, i.e. student, expert and tutor model, remain unchanged. For hypertext systems better link management can be introduced by using a link database which is queried by the `GET` method. These examples just represent a minor aspect of the possibilities CGI offers.

⊕ **Student modelling**
CGI applications reside on a world–wide web server, and in general the possibilities of reading, writing and executing files are not restricted (depending on the operating system and the system developer's access rights). Consequently, modelling the student's knowledge is not limited by restrictions which are imposed by other techniques. In addition, a student's model, either for the current session or for an inter–session profile, can easily be stored: for example in an external database that is queried by the CGI program.

⊕ **Separation of concerns**
It is possible with CGI programs to separate the document and exercise texts from the program logic of a tutorial system. Thus, changes to texts can be quickly made without the need to modify the source code of the system itself. I implemented a similar concept for the Tootsie Development System (see Section 4.2), which is easily localized for different computer environments by changing the resource variables, and not the source code of the CGI programs. CL–HTTP, plug–ins, and Java offer the same possibility, whereas JavaScript requires that the source code of functions is incorporated in a world–wide web document[37].

⊖ **Local**
CGI applications do not offer the possibility of running locally on a student's computer, so an internet connection is necessary when accessing a tutorial system. This problem can be solved by installing a local world–wide web server on each user's computer, but this is not feasable. Consequently, with the need to communicate with many different users the work–load of the server will increase (as mentioned on page 41).

⊖ **Interactivity**
The CGI interface uses a request–response communication, which does not make real interaction between student and system possible. Students must select a `submit` button before data is sent to the CGI program (see also page 41). With a slow internet connection this will effect the usability of a system as the response time will be too long. A suitable solution is to implement plug–ins, JavaScript code, or Java applets.

⊖ **Programming experience**
Writing CGI applications normally requires advanced programming experiencies.

⊖ **Exercise–specific reactions**
As described in "separation of concerns" the exercise texts and the source code of a tutorial system can be stored in different files. However, if system reactions to users' activities must specifically be defined and if the source code is not available (as in many commercial products), the tutorial system developer will depend solely on the implementation of the CGI program, whose programmer hopefully foresaw all the situations which could happen. This problem can be solved if a "scripting language", like Macromedia's Lingo, is included, but it should offer advanced programming constructs which can describe the interaction between system and student. In implementational techniques, which include system reactions in exercise files (like JavaScript), modifications can instead be made by adding the required functionality to the individual files. The advantage is that the same programming language in which the system is written is used, but program logic and document contents are then not separate.

3.5 JavaScript and Cookies

Basically standard HTML documents, i.e. documents that can be rendered by most world–wide web browsers, have static contents, that means that for all users exactly the same information is displayed.

[37] The current JavaScript 1.1 however allows to specify an external source code file which is loaded separately.

However, many world–wide web services, as well as many web publishers, require the possibility of creating the contents of web pages dynamically: document texts can depend on the date or time of the user's access, frequent visitors want to specify certain topics in which they are mainly interested, or more than just one document window must be controlled simultaneously. The already existing standard for dynamic contents, the Common Gateway Interface, does often not fulfil these needs or expectations: firstly, their execution requires valuable server time, and secondly, their flexibility is limited: contents that must continously be updated on a web page can hardly be implemented. With the dawn of Java and JavaScript, which have both been included in the most popular world–wide web browsers by Microsoft and Netscape, these limitations are no longer valid. In this chapter, I discuss the use of the scripting language JavaScript, whereas the more powerful, but also more complex, programming language Java is explained in detail in Section 3.6.1. In addition, I also mention a new method of storing information persistently on a client's hard disk, the so called "Cookies". The handling of Cookies has largely benefited by its implementation into JavaScript. In Section 3.5.3 on "LiveConnect" a way of communication between JavaScript, Java and plug–ins is introduced, before I conclude with a few examples that use the new means for dynamic web contents.

3.5.1 JavaScript Basics

JavaScript was first introduced for world–wide web browsers by Netscape Communications in the years 1995/96. By now the newest release has already reached version number 1.1, and, in addition, Netscape's strongest competitor Microsoft — together they control approximately 80% of the world–wide web browser market — has implemented the similar, but not entirely equivalent, JScript. The potential user group of JavaScript is therefore large enough to be considered as a technology for implementing tutorial systems. JavaScript itself is called a scripting language, and, despite its name, not comparable to Java at all. In general, scripting languages are part of various applications and provide a simple way to control the application with the help of a small program, for example a macro, for tasks that frequently occur. Their main idea is to address more users than the normal programming languages do, by reducing complexity or by making them more alike to a natural language:

> The JavaScript language resembles Java, but without Java's static typing and strong type checking. JavaScript supports most of Java's expression syntax and basic control flow constructs. In contrast to Java's compile–time system of classes built by declarations, JavaScript supports a run–time system based on a small number of data types representing numeric, Boolean, and string values. JavaScript has a simple instance–based object model that still provides significant capabilities [Netscape Developer, 1997a].

JavaScript code is part of the HTML page that uses it, and so it is also called "inline code". However, it can also be stored in a separate file and included by any HTML document, a method which makes handling and maintenance of the source easier. At the time of writing this operation is only available from JavaScript 1.1 on, and is not supported by Microsoft's JScript or older world–wide web browsers. Despite the common impression of scripting languages, i.e. low complexity in favour for ease–of–use, JavaScript supports all basic programming constructs like recursion. Beginners with no programming experience must definitely invest some time before they can start writing code, but the same is also necessary for programming languages that are part of courseware systems, like Macromedia's Lingo. There, the integrated and GUI control–oriented development environment will leverage the coding, but such a tool is also planned for JavaScript. What makes JavaScript easy to use is not only the close resemblance to Java, and the programming languages C and C++, but also:

- **Variable declaration**
  In JavaScript variables are loosly–typed, i.e. the developer does not have to specify a certain variable type, like `char` or `int` in the related programming language C: he just assigns a value to the variable. During the execution of JavaScript programs it does not even matter whether the type of the assigned value remains the same. When necessary, JavaScript will perform type conversions

automatically. By default, variables are defined globally, and only for local declarations the keyword `var` must be used. For beginners, these two features make JavaScript easier to handle, because to a programmer the name or value of a variable is more relevant than its type or location. However, this also means that larger JavaScript applications are more difficult to maintain: variable declarations can appear anywhere in the source code, and variable assignments may even intersect. Therefore I recommend that global variables are defined only when necessary, and variable names are chosen that clearly express the intended use.

- **Objects and classes**
  In JavaScript the commonly known structures, array and record, are in fact not part of the language syntax, they are seen as user–defined class objects[38]. Classes are simply declared by writing a JavaScript function, which is named after the class and parameterizes the class properties. Consequently, if the developer wants to create an instance of a class, he will have to use the keyword `new`, followed by the function call, to reserve enough space for the class object. More complex structures can also contain their own functions, which are known as methods, but unfortunately more advanced techniques of object–oriented programming, e.g. inheritance, are not available in JavaScript. The most frequently used standard classes are `document`, `window` and `navigator`. For example, the first one contains all the properties of the current HTML page, but also gives access to the links and form elements of that page. These can then be altered by JavaScript commands and functions. New classes cannot be derived from standard classes, so the class hierarchy is known as an instance hierarchie (see [Netscape Developer, 1997a]), because objects and not the class itself are available to the developer.

JavaScript functions are normally located in the `<HEAD>` section of an HTML file, so they are loaded before any data is displayed or user events occur (see [Netscape Developer, 1997a]). Otherwise, there are no restrictions in positioning JavaScript code in an HTML file, so whenever the web browser is executing a method or function call, the results will be rendered within the normal document text flow. This just works once while loading a page, but nevertheless, the system developer is able to define dynamic, often user–specific, contents, that depend on external influences like Cookies or additional information on the web client. In contrast to that, event handlers can be triggered by the user at any time while interacting with the web browser. Still, once the document text has been fully displayed in a browser window, it cannot be changed by a function call anymore. What event handlers can do though is manipulate the contents of form controls, do computation, and display a dialog box or another HTML page in the same or new browser window. If I examine the method, in which JavaScript functions are executed by the world–wide web browser, I find a contradiction in the official documentation: in general, JavaScript is *interpreted* by the client, but according to the description of the new object `Function` "[...] declared functions are *compiled*". Except for performance reasons the distinction between compiled and interpreted code will not influence the development process of a tutorial system. The predefined JavaScript methods mainly offer link management, string manipulation, mathematical functions and web browser control. A good overview of all available methods, properties and event handlers — examples for the last two will be dicussed later — can be found in [Netscape Developer, 1997]. As a typical example for the use of a JavaScript function in a tutorial system, `SetTimeout` offers the ability to run a given code expression after a certain amount of time, so for instance, by calling `SetTimeout` recursively the value of a form control or a variable can be continuously updated. In a tutorial system `SetTimeout` may be used to display a help screen after an elapsed time period, or to collect user data by storing how long a student has spent on a certain task.

Amongst the useful properties of class objects for educational programs are `document.cookie` and `document.referrer`. The first one allows to easily set and retrieve Cookie values as described in Section 3.5.2. As in the tutorial system prototype Tootsie — see Section 4.2.4.1 — Cookies can be applied to store student data, e.g. what exercises he has done and what difficulty level he has chosen, in order to select further tasks and feedback individually. In `document.referrer` the URL is kept, from which the current world–wide web page was accessed. This reference may be checked to see whether the student is

[38] Netscape's JavaScript documentation, see [Netscape Developer, 1997a], uses the term "object" differently than I do: whereas I use the commonly known expressions from object–oriented programming, the documentation replaces "class" with "object", and "object" with "object instance". Although strictly speaking JavaScript is not an object–oriented language, the object–oriented terms seem more appropriate to me.

coming from a preliminary page, and following the suggested course flow. However, the most important advantage of JavaScript are the event handlers, which allow interactive HTML pages. Event handlers are regular functions, that are called whenever a user event[39] occurs. Most form controls[40], as well as the web browser itself, support these, so the way in which the user is interacting with the current document, can be followed and stored. Event handlers are mainly applied when simple computations or operations must be done, or fill–in forms are checked for proper input before submitting them to the server in order to reduce the server's work load. However, their abilities can be ideally used in a tutorial system to establish an student profile: e.g. in which order were the exercises done? How many times has the help button been clicked? How often did the student's mouse pointer leave the current browser window? As the tutorial system prototype Tootsie shows, it is also possible to do all the student modelling with the help of event handlers and JavaScript, though the resulting model is very limited. However, in this case the main benefit is that the tutorial system can also run without a permanent internet connection, as no external programs like Common Gateway Interface applications are required. An overview of existing and proposed events is found in the new HTML 4.0 specification, which was introduced in Section 3.1.3. HTML 4.0 acknowledges the use of scripting languages, and points out that "HTML's support for scripts is independent of the scripting language [Raggett et al., 1997]", so alternatives to JavaScript may be announced sooner or later.

The benefit of JavaScript is what [Raggett et al., 1997] calls "smart forms". Before an HTML form is submitted to a server its contents can be checked for erroneous input, thus informing the user immediately and reducing the server's work–load. In addition, networked applications can be build with the help of dynamic contents. However, JavaScript is not standarized yet, and so it is difficult to write universally understood programs, because browsers may react differently to the same code (see [December, 1997]). Improvements have been made up to now, but still JavaScript is not fully compatible. While writing Tootsie I noticed for example that certain commands do not work on all operating systems, even if the world–wide web browser came from the same manufacturer. In particular Microsoft's Internet Explorer 3.x had difficulties with JavaScript code: even on Microsoft's internet sites runtime–errors occurred, and in particular, the code for changing the contents of a frame, which was mentioned in the Frequently Asked Questions list by Microsoft, did not work. However, as long as standard commands and event handlers are used, the compatibility problems can be reduced, as many internet sites that run JavaScript programs show. The developer must remember though that the potential user group of a tutorial system is limited to the web browsers which understand JavaScript, and that tests on different platforms may be necessary.

3.5.2 Cookies

Cookies allow world–wide web servers to store small pieces of information on the client's side, and their contents can be retrieved any time in future connections. In fact, they are not restricted to the scripting language JavaScript only; they are actually part of the HTTP header of any HTTP object, so for instance, Common Gateway Interface programs are also able to work with Cookies. Nevertheless, I discuss the Cookie spezification in the context of JavaScript because of the easy–to–use mechanism that JavaScript provides: a Cookie is a property of the class `document`, and it can be accessed like any string object. As previously stated, Cookies are a persistent piece of information, which is set by the server and kept on the client's side. In general, their life–span lasts longer than the connection to the world–wide web server, so whenever a user returns to the same web site again, the Cookies are still valid, and their contents can be retrieved. They will only be deleted, when their expiration date is reached, or when the user himself removes them from the hard disk. The spezification defines the following parameters for the Cookie HTTP header:

- `<name>=<value>`
  After the Cookie has been set by the server, it can be accessed under the given `name` in order to retrieve its value. A new value is stored whenever it is assigned to the Cookie again. The `name` and `value` sequences themselves contain any character excluding semi–colon, comma and white–space.

[39] A user event is e.g. a mouse click or a keyboard input.
[40] In this context, the expression "intrinsic event" is common in various documentations like [Raggett et al., 1997].

If the length of a Cookie exceeds the maximum 4 kB the additional characters will be removed before the Cookie is set.

- `expires=<date>`
  The expiration date specifies how long a Cookie is stored on the client's side. If no expiration date is given, the current Cookie will be deleted after the user session ends. The same will happen, if the date is set to a value in the past. In any case, the path and the name must exactly match in order to replace the old Cookie with a new Cookie. If the maximum number of stored Cookies[41] is reached, the least recently used Cookie will be deleted, although it has not expired yet.

- `domain=<domain>`
  In order to prevent Cookies being accessed by unauthorized servers, each retrieval of a Cookie value must be confirmed by a "tail–match"[42] of the domain name. The default value of `domain` is the host name of the server, which generates the Cookie.

- `path=<path>`
  "The path attribute is used to specify the subset of URLs in a domain for which the Cookie is valid ([Netscape Developer, 1997e])". By default, the path of the current document will be assigned to `path`.

- `secure`
  The Cookie will only be transmitted if the communication between server and client is secure.

Currently most world–wide web users view Cookies, which are able to store any textual information on the users' hard disk, with suspicion. They fear not only possible security leaks through implementational errors, but mainly the ability to create a profound user profile. Although in most browsers a dialog box is optionally displayed, in which the user is asked to confirm a Cookie whenever it is transmitted, the user must have the opportunity to see and control the contents of Cookies himself. In a tutorial system the student will be able then to try various "what–if" situations by modifying the Cookie values. In Tootsie Cookies are frequently used to learn more about the student, to give feedback, and to set user preferences. The tool "Cookie cutter" helps the student to check the values stored in these Cookies, or to change their contents. By making Cookies accessible to the students, I therefore hope that the user of a tutorial system will accept and trust them. At the time of writing the prototype Tootsie is the only educational system that uses JavaScript in conjunction with Cookies. The lack of experience and examples in regard to the implemention of tutorial systems in JavaScript was the main reason for writing the Tootsie system. In addition, the prospect that Cookies work without a permanent internet connection, for instance when running on a student's computer at home, was a compelling argument for the technique JavaScript in the software design process[43]. Other implementations of Cookies include shopping in the world–wide web, or storing login and connection parameters. In the first example, the Cookie works like a shopping cart, in which the user puts goods, before he pays at the check–out. The second use is applied by the PUSH system (see Sections 3.4.2.3 and 3.6.2.1): it works with Cookies to make connection handling between the client and the Prolog process on the server easier. The socket number, which each user gets when accessing the Prolog database for the first time, is stored in a Cookie in order to be sent to database when a new queries must be processed.

3.5.3 LiveConnect

The techniques that are suggested in this thesis can be combined to enhance the abilities of a world–wide web based tutorial system. If a developer uses cascading style sheets for the layout, the scripting language JavaScript for checking user input and setting preferences, and finally Java or plug–ins for simulations and interactive graphs, he will also require a way to communicate with the different objects in a web page. In

[41] This may differ from client to client, however the minimum numbers are: 300 Cookies in total, and 20 Cookies per server or domain.

[42] "Tail matching means that [the] domain attribute is matched against the tail of the fully qualified domain name of the host ([Netscape Developer, 1997e])".

[43] As far as I know only Netscape's Navigator supports this functionality. For Microsoft's Internet Explorer the non–existing internet connection could be replaced by the Personal Web Server.

this case, he will need Netscape's LiveConnect to "call Java methods from plug–ins, call native methods implemented in plug–ins from Java, call Java methods from JavaScript, [and] call JavaScript from Java methods ([Netscape Developer, 1997d])". The same functionality is also available for Microsoft's Internet Explorer, so the possibility of connecting HTML, plug–ins, JavaScript and Java is not limited to one browser manufacturer only. According to the LiveConnect documentation JavaScript is now part of the Java environment, so every public class of Java can be accessed by JavaScript. As Java applets are embedded into a world–wide web document, they can be referenced with JavaScript by their given applet name in the `document` object. If a method call to a public function of a Java class is added to that applet reference, the method will be executed. Plug–ins and Java classes interact with the help of the Java Runtime Interface JRI. Plug–ins can define public Java classes that are initiated at the same time whenever a plug–in is executed. As mentioned before one benefit of JavaScript is that all public Java classes, which are currently loaded, can be accessed by a script, so as a result JavaScript is able to communicate with the classes of the plug–in as well. In addition, JRI allows Java classes to call native[44] plug–in functions (see [Netscape Developer, 1997d]).

3.5.4 Discussion

JavaScript only operates within an HTML document, so the implementation of exploratory learning environments is possible. In addition, JavaScript also allows adaptivity, student monitoring and guided–discovery learning on a basic level, as event handling, Cookies, and individual user support are part of its functionalities.

⊕ **Availability**
JavaScript can already be applied in current world–wide web browsers, i.e. Netscape's Navigator and Microsoft's Internet Explorer. The latter however, does not implement all the classes and class methods of JavaScript 1.1.

⊕ **Portability**
JavaScript is platform–independent if a JavaScript capable browser exists for a particular computer architecture or operating system. The only exception is the newline character in a `textarea` object: for Unix and Apple Macintosh platforms `\n` is used, while Windows encodes newline as `\r\n`. However, the author of a world–wide web document which incorporates JavaScript can query the user's platform with the `userAgent` property of the `navigator` class.

⊕ **Integration**
As JavaScript must be part of an HTML document it can be used in conjunction with other techniques: LiveConnect can establish a communication between Java applets and JavaScript, while Macromedia's Authorware can directly execute JavaScript functions with the procedure `GoToNetPage`. According to [Mallery, 1997] the CL–HTTP server also supports JavaScript.

⊕ **Extension**
The scripting language JavaScript extends standard HTML by the ability to process user input and data. Before information is sent to the world–wide web server, forms can be checked for erroneous input and basic computations can be made. In contrast to CGI, the browser itself handles these requests, thus reducing the work–load of the server. In addition, individual document texts can be presented to the students and depending on the students' access history the course flow can branch with the help of JavaScript's `if-else` command.

⊕ **Interactivity**
JavaScript does not offer the possibilities of Java or plug–ins, but its event handlers can provide feedback in many situations: in JavaScript 1.1 for example, images can be changed even after the world–wide web browser has rendered them by calling a function, which replaces the existing image with the new one, for the event `mouseOver`[45] of an image object. However, it is still not possible to change the document text after it has been displayed. In addition, events can be used to monitor the student's activities.

[44] A plug–in is a native–code extension of a world–wide web browser.
[45] This event is triggered whenever the user's mouse pointer is over an object which supports the event.

⊕ **Local**
JavaScript code is executed on a local computer, thus not requiring an internet connection. With Netscape's Navigator it is also possible to set and read Cookie values without an HTTP server, however Microsoft's Internet Explorer 3.x does not support this functionality.

⊖ **Standard**
JavaScript was solely developed by Netscape and is currently not standarised by an independent institution. Although Microsoft has included most of JavaScript's features into its world–wide web browser, the problem of proprietary commands is still not solved.

⊖ **Modular**
As JavaScript combines exercise texts and source code in one file a tutorial system can hardly be modularised, for example in separate student, tutor, and expert models. Consequently, JavaScript should only be used for a basic adaption or student monitoring procedure rather than for complex student modelling as demanded by intelligent tutoring systems.

⊖ **Implementation**
In general, JavaScript is a well–defined scripting language which is already used by many world–wide web sites. However, I had various problems while writing the tutorial system and development toolset Tootsie: for example, a system which was running with one version of the Netscape Navigator did not operate with the next release. Erroneous implementations of the JavaScript interpreter unfortunately hinder the development of JavaScript based world–wide web documents. Thorough tests on various platforms are therefore inevitable.

⊖ **Programming experience**
Development of JavaScript applications requires basic programming experience.

⊖ **Document size**
If JavaScript is used for complex data processing and computation, the size of a world–wide web document will grow because the source code is included in a document. With slow internet communication facilities this can affect the usability of a system, so the complexity of a tutorial system is restricted. In JavaScript 1.1 a source file, which is stored and loaded separately, can be defined, however this functionality requires a JavaScript 1.1–enabled browser.

⊖ **Cookie acceptance**
Many world–wide web users refuse to store Cookies on their hard disk or in the home directory. However, tutorial systems often require personalised data which was collected from the student during more than one session in order to follow the learner's progress. The use of Cookies should therefore be evident to the students.

⊖ **Printing documents**
Document text which is displayed on screen by using the `write` method cannot be printed. Therefore, `write` should only be applied for contents which are individually chosen for a student.

3.6 Java

All the previously mentioned techniques had to rely on HTML and the accompaning web browser for presentation and operation in the world–wide web. CL–HTTP, plug–ins and JavaScript, all use either HTML and the world–wide web browser as their graphical interface or are embedded into an HTML document. The programming language Java on the other hand is able to run independently of the web browser and HTML, but still offers the ability to establish data connections to the internet. In the following sections I will introduce the programming language Java, whose revolutionary concepts have greatly influenced the development of the world–wide web over the last three years. As before, I will mainly concentrate on the advantages and disadvantages in regard to tutorial systems, and I will also discuss two examples, that already use Java in educational software.

3.6.1 Java Basics

The programming language Java was invented by James Gosling at Sun in 1990, when he and his team were looking for a language that was more appropriate for writing consumer electronics software than the existing languages C and C++. At that time C++ was not yet standardised, while C was missing the modular and object–oriented program design of C++. In addition, it is still necessary to recompile applications that are written in C or C++ when transfering them to another system platform. The developers of Java hope now that all these disadvantages will be resolved, because Java is a small, reliable, architecture–independent language, that is already prepared for internet programming, because methods for accessing the world–wide web are included in the API[46] package by default. Java programs are available in two forms: either as a stand–alone application or as a Java applet. The latter only runs in the context of a world–wide web browser, and therefore it is rendered as an inline object in the browser window. Java applets extend the possibilities of web browsers as well as plug–ins do, but as a virtual Java machine is part of modern web browsers, they do not have to be separately installed, and their usage is hardly limited, especially not to a certain user group. According to [Flanagan, 1996] Java combines the following characteristics:

- **Simple**
  For most programmers learning Java is easy, because its syntax and code structure is closely related to C and C++. In addition, some of the more difficult features of C and C++ have been removed, like operator overloading, pointers, or the compiler and link control of the C preprocessor section. However, an absolute beginner in programming, which many content providers of tutorial systems certainly are, will still have his problems in mastering Java. Consequently, the lack of time for learning Java will restrict its usage as a core technology of tutorial systems amongst unskilled system developers. However, this should not be the sole reason to reject Java, as its inherent internet functionality and its possibilities of direct user manipulation can hardly be replaced by another technology.

- **Object–oriented**
  In general, the object–oriented programming model, which is more or less used by HTML 4.0, Common Lisp, and the courseware applications as well, promises modular and reusable code, which was one of the premises of Java. In contrast to C++, Java has already incorporated the object–oriented approach: most "commands" are actually methods of classes. At first, programmers who are accustomed to C may have difficulties in adopting the new way of writing codes, but they will soon realize that object–oriented programming is often closer to reality.

- **Distributed**
  Java contains class methods for connecting to a socket (to establish a reliable stream network) or accessing a internet resource by an URL. However, it does not offer services like CORBA[47], as this would go beyond the scope of Java. For a tutorial system the available methods certainly cover all required communication needs, but it is advisable to consider implementing a link management facility, which the world–wide web does not offer.

- **Interpreted**
  The Java compiler generates platform–independent byte–code[48], which is then executed by a platform–specific interpreter and run–time system on the client's machine. As the advantages of Common Lisp (page 28) have already shown, an interpreted language like Java enables rapid prototyping and easy experimentation, because compiling the code and linking the libraries is not necessary anymore. However, due to performance reasons many environments that run Java applications include a just–in–time compiler as well, which translates the byte code into native machine code. For a tutorial system however, there should be no performance difference between interpreted and compiled Java code.

- **Robust**
  Due to automatic garbage collection, exception handling, and the lack of pointers, Java development

[46] **A**pplication **P**rogramming **I**nterface.
[47] **C**ommon **O**bject **R**equest **B**roker **A**rchitecture.
[48] Unfortunately "[...] the byte–code is still partly erroneous (p76, [Sackl, 1997])".

is more robust than C or C++, but still the programmer is responsible for writing software that is easy to maintain and stable. A tutorial system especially requires a robust implementation, as user acceptance largly depends on an error–free learning environment. The importance of a robust implementation can be seen in software development where it is common to plan a third of the project time for defining and designing the project, and another third for testing. The remaining time is then used for coding and installing the system.

- **Secure**
  Unfortunately the missing organisational structure of the internet makes abuses very simple: data can be collected, modified, or replaced at any node of the network. As Java is executed on a local machine, one of the key design issues was to make downloading and running Java applets as secure as possible. A byte–code verification process is used to prevent illegal code, stack overflow (or underflow), incorrect register operations, or illegal data type conversions. In addition, with a separate name space for downloaded classes it is ensured that standard Java classes cannot be overwritten or replaced. For security reasons and in contrast to stand–alone programs, Java applets do not have the right to write on the user's hard disk. Therefore, [Sackl, 1997] rejects applets in his prototypal implementation of a communication tool for workgroups, and uses a Java program instead. With the introduction of "trusted applets"[49] however, which have the same rights as local Java programs, the known user interface of a world–wide web browser does not have to be replaced by a proprietary interface which is required when using a Java program, and specific user data can be stored and retrieved on the user's computer, thus relieving the server from managing various students' report or log files. According to [Sun Microsystems, 1997] the later releases of the trusted applet spezification will provide more sophisticated security policies, including greater granularity in the allowable trust levels.

- **Architecture–neutral**
  By using Java software developers hope that one day it will be possible to write programs that can be transfered between to different computer platforms without modifying the source or executable file. According to [Flanagan, 1996] the internet will therefore become the computer itself as the never–ending discussion on the network computer promises. Thanks to the byte–code format Java programs can run on any computer platform as long as a Java interpreter and run–time system exist. For software developers this is very important as it no longer limits their programs to just one platform. Instead, their applications can now be distributed for a broader user group without any modifications. Like CLIM Java uses an abstract windowing toolkit (AWT) to design graphical user interfaces for platform–independent Java programs. This toolkit must have the ability to "adapt" to the current user environment automatically. Unfortunately this technique coincides with the restriction of forbidding platform–specific window operations: the smallest set of window functions, that are common in all the different user interfaces, can be used. In Java itself, platform–independent graphical user interface is supported by creating platform–dependent "peers" for each of the classes and components of the abstract windowing toolkit ([Flanagan, 1996], p269). With "Swing" Sun has recently introduced a new extension to the AWT which offers two alternatives: a Java program either provides the *same* look–and–feel as any other application for a particular computer, or it consists of special cross–platform components that are equally presented, no matter what operating system they are running on (see [Sun Microsystems, 1998a]).

- **Portable**
  In contrast to C and C++ the sizes of data types are exactly specified in Java, as well as the operations that can be used for a single data type. This also limits the possibilities of errors, and increases the chances of a architecture–neutral implementation.

- **High–performance**
  Although just–in–time compilers already exist, Java is mainly an interpreted language. Therefore, it is 20 times slower than C/C++–compiled programs, but according to [Flanagan, 1996] faster than scripting languages like Perl. For tutorial systems especially the performance of the underlying

[49] With the help of encryption technology "[...] it is possible to load a trusted applet (one that can run without severe security restrictions) over an untrusted network as long as you trust the source of the applet (p200, [Flanagan, 1996])". In general, applets "[...] loaded into a Java–enabled browser cannot read [write] files. Sun's appletviewer allows applets to read [write] files that are named on the access control list for reading [writing]. [...] However, an applet can maintain its own persistent state on the server side ([Sun Microsystems, 1998b]).

programming language is not important, because these systems are waiting for a student's reaction most of the time, and thanks to Java's multithreaded design, processor resources are not wasted, when, for example, waiting for an internet connection. However, if many computations are necessary and the application itself does not require world–wide web accessability, then depending on the speed of Java just–in–time compilers, a different programming language will be more feasable.

- **Multithreaded**
  Java supports the use of multiple threads[50], which can be executed simultaneously, by offering their own class definition. Synchronization primitives are also included to handle the accesses to mutual system resources, which must be used exclusivly by the different threads. They are based on the monitor variable concept, which was introduced by C.A.R. Hoare in order to replace Dijkstra's semaphores which cannot prevent certain situations leading to deadlock or starvation. Multithreaded tutorial systems improve the performance by the aforementioned technique of doing further computation while downloading, or even pre–loading, the following exercise page. Therefore, a tutorial system developer should apply multiple threads to make the response time of the system as short as possible. However, implementing this technique will require advanced programming skills.

- **Dynamic**
  "[...] Java loads in classes as they are needed, even from across a network. Classes in Java also have a run–time representation. Unlike in C or C++, if your program is handed an object, it can find out what class it belongs to by checking the run–time type information. The run–time class definitions in Java make it possible to dynamically link classes into a running system ([Flanagan, 1996], p9)."

[Sackl, 1997] sees the disadvantages in Java not primarily in the programming language itself, but first and foremost in the support for developers. Java is a fairly new language, which must still prove whether it can fulfill the high expectations that are set into it. In the beginning professional development toolsets were rare, and so other languages were prefered, but this has now changed. In addition, the new Java Development Kit by Sun provides extended features, that have been incorporated in regard to the experiences that were made by developing applets for the world–wide web. So, JDBC[51] was introduced to enable Java to execute SQL statements, and JDBC continues the tradition of the first Java specification to be portable, that means that JDBC is not restricted to one database only. JDBC establishes a connection with a database, sends SQL statements, and processes the results, thus offering the abilities of standard databases to any Java application. For instance, tutorial systems could use a database to store user models or exercises, to query the students' progress or to keep the exercise texts independent and modifiable from the system code. In comparison to other world–wide web techniques Java offers real interactivity for the user. [Anderson et al., 1995] summarizes the requirements for the interaction with a interface:

- "Actions taken to the interface must be passed through the tutor. The tutor needs to know what actions students have taken so it can follow students along the solution path they are pursuing and provide appropriate guidance."
- "The tutor must be informed about the consequences of any interface actions for the state of the interface. Basically, the cognitive model needs to maintain in its working memory a representation of the interface that the students see."
- "The tutor must be able to perform interface actions itself."

Like in plug–ins, user events will only be triggered if they happen in the content area of the applet, or if they have been passed on by LiveConnect. However, the aforementioned requirements are still best achieved with the help of Java: like plug–ins the contents of an embedded applet can be continously changed as an applet is independent from its underlying browser, but although plug–ins offer the same level of interaction, they do not have the possibility of keeping a connection to the server constantly open. If the tutorial system is realized with the help of a database, this difference will be an especially

[50] Threads are lightweight processes that do not require the organisational overhead of standard processes. Often threads must share the same system resources, while processes can rely on their own memory space etc.
[51] It is thought of standing for **J**ava **D**ata**b**ase **C**onnectivity.

important argument for the use of Java. In all other techniques, i.e. HTML, CL–HTTP, CGI programs and JavaScript, the content of a world–wide web page cannot be modified, once it has been rendered by the web browser. The only exceptions are JavaScript event handlers, which can display dialog boxes or change the values of form controls, and dynamic objects, which for example will be able to change their position in a page. For further information and, for the newest Java spezification I recommend the documents by [Sun Microsystems, 1997] be frequently read.

3.6.2 Examples for Tutorial Systems in Java

For a tutorial system in the world–wide web the use of Java is recommended if complex programs must be realized. The following examples will show what benefits Java has for educational software: previously unthinkable types of tutorial systems, like simulations, can now be made available in the world–wide web. In addition, Java applets can be combined with all other techniques to implement a more intuitive user interface. At the time of writing I do not know of a tutorial system, which was solely programmed in Java as a stand–alone application, but many examples of Java applets can be found. Although programmers have focused on applets for special textual or graphical effects, the number of educational applets is growing, and can be accessed from the Java applet repository at Gamelan[52].

3.6.2.1 Example: PUSH Graphical User Interface

PUSH, a document retrieval system for SDP manuals, has already been introduced in Section 3.4.2.3, where I have focused on the CGI programs that are necessary to access the underlying Prolog database. Retrieving SDP manuals is also possible by using an interactive graph, whose document nodes are organised in a hierarchical tree structure which contains all the relevant dependencies. Direct user manipulation, which enables the user to browse the information domain by clicking on the various graph nodes, and creating graphs on–the–fly, whenever links are selected in the text document beneath, cannot be implemented as a CGI program. Therefore, the programming language Java was chosen (see [Espinoza, 1996]). Firstly the Java applet is intended to give the student orientational guidance in the information space, and secondly it is an alternate way to access the manuals. The data, which the student is interested in, can be changed either by selecting a link in the document text or by clicking on its represented object in the drawing area of the applet. The focused object is then automatically centered in the graph and neighbouring objects and relationships are added. For beginners, this option provides an easy way to browse the information domain without losing orientation, a problem which frequently arises in unstructured hypertext environments. The graph applet is displayed in a separate frame of the browser window to make it independent from the remaining text, and to reduce download times as the applet must just once be loaded into memory. The graph data itself is provided by the Prolog database, whose query results are written to a file that is remotely read by the Java applet: "The reason that a socket connection[53] is not used is that at the time of implementation (fall 1995), the Netscape browser was not equipped with this functionality ([Espinoza, 1996])". The PUSH interface is interactive on several levels, mainly because the user's ability to understand new information depends not only on the user's previous knowledge, but also on the spatial representation and accessability of the domain. By offering the possibilities to retrieve SDP manuals in different ways, i.e. menus and graphs, the various user preferences can be satisfied (see [Espinoza & Höök, 1996]). In an evaluation test two versions of the PUSH system were examined by Kristina Höök: the adaptive one offered the same features as have been mentioned here and in Section 3.4.2.3. The non–adaptive interface was very similar to the other system in the test, except that all the information entities of a new manual page were closed when it was displayed for the first time. Höök found that the adaptive system was prefered by the students, but she correctly remarks that such comparisons can be questioned as the underlying design of the test systems is different, thus influencing the test results (see [Espinoza, 1996]). However, in that test the authors believe that they found a way to prevent that obstacle as the non–adaptive version is also "a good system in itself". I do not agree with the last statement: as I said before the document text was collapsed when it was presented to the student the first time in order to prevent an information overflow. Consequently, the student had to extend each topic in which he was interested, so more user actions

[52] http://www.gamelan.com/.

[53] The Common Gateway Interface programs communicate by a socket with the database, see page 49.

were necessary and information could be missed. This certainly modified the preconditions of the evaluation and thus the outcome of the test as well, but the benefits of adaption remain undisputed nevertheless.

3.6.2.2 Example: Powersim Simulations

Powersim is a Norwegian software company that produces development toolkits for world–wide web based simulation programs. The software package Powersim Metro JX Suite contains a tool for building a simulation, called Powersim Constructor, and a simulation server Powersim Metro Server, which is responsible for the communication and interaction with the clients. The clients themselves are world–wide web browsers that must be able to interpret Java code, because the user interface of a simulation is realized as a Java applet (alternatively, Microsoft's proprietary ActiveX can be integrated). The main intentions for writing the software were that the effects of future changes to a running system can be visualized and tested beforehand. Various scenarios can be examined and new strategies tried before an organisation is fundamentally restructured. A simulation is based on a network structure consisting of elements and links, whose overall concept was adopted from the "system dynamics model" which was developed at the M.I.T. Elements are described by mathematical equations, and according to these their behaviour is simulated by the system. Constructor supports the developer in building a system: it provides a drag–and–drop user interface, that can also integrate video and audio files into a simulation, and contains standard elements, which are known from the "system dynamics model". The Metro Server is responsible for the communication and synchronization between the client's user interface and the server's simulation engine. A persistent data stream is kept open during a user's session, so it appears to the user that the simulation is running locally, while the server is doing all the calculations. In this respect Java is needed for establishing the connection as well as for the design of an interactive graphical interface, because the results of the simulation, whose changes are transmitted from the server to the client, must be displayed. The benefit of a server–based simulation is that the user's interactions can be continously monitored and analysed by the developer, as data must be written on the server's hard disk due to the aforementioned applet security scheme. The Metro Server supports up to 100 clients simultaneously, allows different roles for the users, and can execute multiple models at the same time. For presentations a simulation can also be started without an internet connection, but especially the global accessibility distinguishes the Powersim products from stand–alone programs. In regard to tutorial system development the simulation software is primarily intended for business adminstration courses, as the examples at Powersim's world–wide web site show. However, other simulations, whose domain can be mathematically described, are possible.

3.6.3 Discussion

In general, all types of tutorial systems can be implemented in Java, as it offers all the possibilities of a programming language. The focus is on simulations and interactive applications, which cannot be supported otherwise: most of the previously mentioned techniques only provide static contents (HTML, CL–HTTP, CGI, JavaScript) in conjunction with basic event handlers (JavaScript).

⊕ **Availability**
Java just–in–time compilers are currently included in most world–wide web browsers, so the user group which can access Java applets is large enough for an implementation of a tutorial system solely in Java. In addition, Sun offers, with the program "appletviewer", the possibility of running Java applets outside a world–wide web browser.

⊕ **Portability**
One of the primary goals of Java is to be portable to various computer platforms. At the moment this ambitious plan has been fulfilled, however the latest arguments between Microsoft and Sun about the Java standard increase worries that proprietary extensions will be made one day.

⊕ **Standard**
At the moment there is one authority which is in charge of the Java standard. System developers can therefore rely on the publications and documentation by Sun.

⊕ **Interactivity**
With its full control over the applet window in a world–wide web browser Java is able to offer real interaction between the system and the user. Various user events can be triggered, so the implementation of simulations and interactive graphs is possible. In addition, Java is a programming language which supports the development of complex applications (in contrast to scripting languages whose abilities are often restricted in favour of easier programming).

⊕ **Local**
Java applets which do not require a permanent internet connection can run locally on the user's computer, and so they do not increase the work–load of world–wide web servers (in contrast to CGI programs). Stand–alone programs are by definition locally executed.

⊕ **World–wide web access**
Java includes classes and methods to establish an internet connection, so in contrast to the aforementioned technologies it is possible to replace the user interface of world–wide web browsers with a more suitable design for a particular course or tutorial system. As Java can communicate with external programs, like databases etc., it also integrates their functionalities, and provides a world–wide web interface for them.

⊖ **Exercise–specific reactions**
If the exercise texts and the source code of a tutorial system are separate from each other (or if the source code is not available like in many commercial systems), it may be necessary to provide a method in which reactions of the tutorial system on the student's activities can individually be specified for an exercise: either the source code of the tutorial system must be modified or a "scripting language" included in which the reaction to various situations can be defined. In general, this is not required if system code and document texts are combined in a single file, like in HTML and JavaScript documents, as the functions, which represent system reactions, can be selected and adapted to the current needs.

⊖ **Programming experience**
A system developer needs advanced programming experiences when using Java.

3.7 Conclusion

A good tutorial system which is implemented on the world–wide web combines the benefits and possibilities which each technique offers. The following enumeration presents ideas for integrating the various technical suggestions which I discussed in this thesis:

- A resource base of exercises and documents, in which the student can discover an extensive area of the course domain (including links to external sources), is connected by the hypertext language HTML, so the previously independent information forms a coherent and exploratory learning environment.
- The use of an external link database, which is best realised with the help of a CGI application, or Hyper–G, is therefore recommended as link management becomes easier and more stable.
- The CL–HTTP server should in particular be considered by tutorial system developers if a knowledge base written in Lisp exists.
- Real interaction between system and students is only possible with plug–ins and preferably Java applets. Within their borders of the browser window they provide the necessary functions to react individually on the users' activities. If the students' problem solving process is monitored inside these embedded objects a global, i.e. not exercise–specific, student model can be created by storing the retrieved data on the server's hard disk. The same can be made for information which is, for example, collected with JavaScript from outside the objects by using LiveConnect. It establishes a communication link between JavaScript and the Java applet, so a JavaScript function can generate a data string from the monitored values which are then transmitted to the applet by calling a Java class method.

- In general, JavaScript is better suited for checking the contents of HTML forms and doing basic computations before data is transmitted to a CGI application or the knowledge base of a CL–HTTP server, but it can also be used to permanently store students' preferences like the design layout of the table of contents (see the explanation of the Cookie variable `tootsie_cont` on page 68).
- If a CGI application is responsible for storing a student model, data of the learner's problem solving can be collected by JavaScript event handlers. Whenever the student solves an exercise this data can be added to the CGI request method.
- Like Java CGI programs are well suited to implement an intelligent tutoring system on the world–wide web. Both can access expert, student, and tutor models which are separately stored from the HTML documents that represent the user interface of an ITS.

These technical recommendations for a good tutorial system depend on the underlying type of system: a simulation requires more interactivity and complex computations than a traditional CAI program, so it is better written in the programming language Java. In general, the separation of the system architecture into user interface, student model, expert model, and tutor model is suggested, however their weights in the overall system structure are subject to the didactic principles: they specify how knowledge is presented, how much feedback is given and how the different motivational dispositions of students are supported. System developers should therefore early involve the intended user group in the design process, in particular for the implementation of an adaption process and a cooperative work environment.

Chapter 4

Implementation

4.1 Tootsie

In the previous chapters I have discussed various techniques which can be currently used to implement an interactive, adaptive, and even intelligent tutorial system in the world–wide web. I discussed their advantages and disadvantages, and presented examples of tutorial systems already in use. Nevertheless, I have not been able to follow the design process of such a tutorial system, partly because some techniques have not yet been tried in real life or their code is copyright protected. I have therefore decided to work on a prototypal implementation of an adaptive tutorial system. The following chapters will cover the design process, give an overview of the system's features, and finally discuss the system's strength and limitations.

4.1.1 Tootsie Basics

4.1.1.1 Tootsie System Components

At the beginning of the design process the main goal was to develop a prototype of a tutorial system to accompany a lecture on Probability Theory and Statistics for computer scientists at Technische Universität München. I quickly found out that working on a tutorial system, especially its exercises, is often a repetitious development process. This means that many development steps occur quite frequently, whereas the core design of a tutorial system is often a unique implementation. Programming exercises for the system and adding them to the existing environment can be tedious, error–prone and time–consuming for a developer who has to do certain steps for each of the exercises again and again. Therefore I decided to split up the tutorial system into a development toolset, which will help in creating exercises, and a user interface, which the student will see (symbolised in figure 4.1). A side–effect of this separation of concerns is that the toolset can now be used more flexibly. The tutorial system is no longer limited to exercises on Statistics; as long as the developer appreciates the restrictions of the technique another course can be designed.

4.1.1.2 Classification of Tootsie

The design process started with an analysis of what the student is supposed to learn during the course. As no preliminary knowledge of Statistics is required, the student will be introduced to the basic principles of probability theory and statistics, for instance Kolmogoroff's axioms, Bayesian laws, and hypothesis tests. In this case drill–and–practice programs are the most suitable choice. The advantages and disadvantages of classical drill–and–practice tutorials have been outlined in Section 2.2.1, and based on these I made the decision to add more functionality to my system in order to compensate for the different needs of the students:

- **Hypertext environment**
  My system is based on the hypertext mark–up language HTML which makes it possible to be accessed not just within the university but also from outside. The user can select a hypertext link

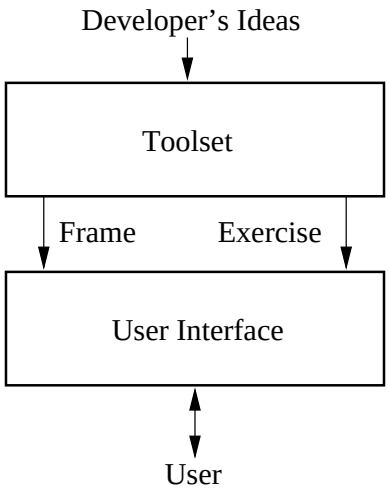


Figure 4.1: Architectural scheme of the Tootsie environment. The system developers enter their exercise ideas into the toolset which creates the appropriate pages and an organisational framework. The pages are displayed in the user interface.

to get more information from the glossary, to ask for help, to choose a different chapter of the course or to change to the cooperative work area of the system. Therefore the often strict linear flow of drill–and–practice programs can be influenced by the students themselves.

- **Table of contents**
  From the table of contents the user will be able to select any desired chapter. As the table of contents is not automatically generated, but manually created by the developer, certain chapters or exercises can be hidden from the user's access. This functionality breaks with the guided tour approach of classical drill–and–practice tutorials.

- **Add–ons**
  A pocket calculator and value tables have been integrated into the system, thus the user does not have to leave the workspace for simple calculations.

- **Cooperative work area**
  The cooperative work area offers a basic chat tool and a discussion group. Here I have tried to create the cooperative workspace which is often demanded by tutorial system theorists. Section 4.3.2 explains their functionality and their intended use in–depth.

- **Exercise wizard**
  In general, the exercise wizard is an overview page detailing what the user has done up–to–now and which exercises are recommended to be visited next. It was introduced to give the user a new starting point when they become "lost–in–hyperspace", a negative side–effect of hypertext systems about which a number of writers have frequently complained. Section 4.3.1.4 gives more details of the information offered by the exercise wizard.

- **Adaption**
  Tootsie offers a simple adaption layer to the developer. According to the user's input the adaption variables will be set and the globally stored information can be accessed by the system. How these values are interpreted by the system and what the reactions are, must be defined and programmed by the developer.

It is not correct to call my system a traditional or even intelligent tutoring program. With the classification of Section 2.2.2 in mind, a traditional tutoring system introduces a complex subject, asks questions of understanding and proceeds differently based on the student's answers. With the introductory course on Statistics I had planned mainly to train the student's abilities to solve exercises in the final exam at the end of the semester, so the complexity of the subjects is low and is covered in the lecture. Additionally, the system architecture is different from intelligent tutoring systems as Section 2.3 shows. Basically it is still

possible to transform my system into a more advanced CAI program by using a more complex teaching scheme, introducing new types of exercises, or adding new adaption variables. The flexibility of my system is therefore discussed in the extra Section 4.2.5. The differences between my system and an intelligent tutorial system cannot be so easily compensated for. As the classification shows, an intelligent tutoring system is based on theories of cognitive psychology and artificial intelligence. Often a knowledge–based system is the fundamental core of the program. The modifications to my system would be complicated and they would not conform to the currently used technique, as accessing external applications is not supported by JavaScript. Java, CGI programs or the CL–HTTP server are therefore better suited to query a knowledge base and store detailed student's profiles.

4.1.1.3 Implementational Technique

What kind of technique are we using? I have decided to base the tutorial system on JavaScript and Cookies. Section 3.5 describes the technique precisely, so I will summarize here the reasons on which my decision was based:

- **Availability**
  JavaScript is part of modern world–wide web browsers, so most users can access documents which contain JavaScript source code.

- **Portability**
  My targeted user group has access to the world–wide web with Netscape Navigator 3.x only, so the difficulties with Microsoft Internet Explorer are not considered as a major problem. Nevertheless it may be necessary to modify my system to ensure that it will also run with Internet Explorer.

- **Integration**
  In world–wide web documents that contain JavaScript code other technologies like plug–ins, CGI program calls, and Java applets can be integrated, so functionalities which JavaScript does not offer are then available. A communication link between the different objects is established with LiveConnect.

- **Local**
  With Netscape's Navigator 3.x students have the possibility to download the tutorial system files and access them from their hard disks.

- **Intended use**
  In general, Tootsie is a drill–and–practice program with additional functionalities such as user adaption (see Section 4.2.4) and the exercise wizard (see Section 4.3.1.4), which could be implemented with JavaScript and Cookies. Complex learning environments however, like intelligent tutoring systems or simulations, require the possibilities which only Java, or CGI and CL–HTTP in conjunction with external programs, can offer.

- **Time–factor**
  The work on this master's thesis was scheduled for a six months period. In that short amount of time JavaScript promised the most and best results as learning a new programming language was not required and as organisational details, for instance setting up a CL–HTTP server, did not influence the start of coding.

- **Novelty**
  At the time of writing I am not aware of a similar system which uses JavaScript for user adaption and course management to the extent that Tootsie does. Consequently, trying and examining the possibilities of JavaScript promised a particular impulse for the on–going work on the thesis. The outlook in Chapter 5 will also suggest further extensions and modifications to my prototypal implementation.

The main reasons for rejecting the other techniques were:

- **HTML 4.0**
  All the proposed HTML 4.0 extensions have not been added to the standard yet, so most of the browsers, which are currently in use, cannot interpret the new HTML tags. Additionally none of the introduced link types provide the functionality to adapt to the user's needs.

- **Knowledge–based HTTP–server**
  The recommended CL–HTTP server shows how easily existing knowledge–based tutorial systems can be published in the world–wide web without any fundamental changes. However, as I had to design a tutorial system from scratch, I did not have a knowledge base on which a CL–HTTP server could be built on. Unfortunately a new knowledge–based system cannot be implemented within a few months.

- **Plug–in**
  External tutorial systems written with development tools like Macromedia Director can only be accessed if the appropriate plug–in is installed. Unfortunatly this is not the case at the computer centre of the Technische Universität München. As students from that university are the targeted user group, plug–ins were rejected at an early stage of the development.

- **CGI**
  The main reasons against CGI applications are the higher work–load on the server, which will result from the number of queries by the different user clients, and the request–response protocol which hinders real interaction between student and system.

- **Java**
  Most of the functionalities, like interactive graphs or basic tools, can also be implemented as Java applets which are then integrated into the tutorial system. Thus, it was not necessary to design a system which was solely coded in Java. In addition, the commonly known user interface of a web browser provides a suitable navigation control of the tutorial system, so enhancements of the interface which can only be realized in Java were not necessary.

4.1.2 Overview

The next chapters describe implementational details of the toolset and the user interface of the tutorial system. I will show what preparation was needed, describe how the toolset is used and explain how exercises, which have been generated by the toolset, can be extended for more user interaction. In the chapter on the user interface I will answer the questions concerning the use of the cooperative work area and how the system assists the student.

4.2 Tootsie Development System

By programming the toolset for the tutorial system Tootsie I avoid making system developers repeatedly work on the same tasks for a number of exercises. For example, JavaScript functions which specify the reactions of the tutorial system on user's input are stored within the HTML file, so it is inevitable that certain code sections appear in many files of the system. If these are manually copied by the system developers the risk of errors is high. The toolset consequently reduces this risk by offering a user interface in which all the relevant data can be entered by the authors.

4.2.1 Preparations

4.2.1.1 Step 1: Technique

Some preparations must be made before the tutorial system toolset can be used. The developers must firstly ask themselves what type of course they want to write. They should carefully consider whether JavaScript and Cookies are appropriate for their needs. In general, this technique is recommended for

exercises which train the students in basic skills, but not for courses which include complex problem solving. A course generated by the Tootsie development system is therefore best combined with an introductory lecture which provides basic knowledge skills needed in a course. I also recommend asking questions of the knowledge domain in the form of multiple–choice exercises, so student monitoring can be based on event handlers.

4.2.1.2 Step 2: Exercises

Once the technical problems have been discussed the developer must prepare the exercises and decide in which order these should be presented to the students. In its current version the toolset offers four types of exercises:

- **Single–choice single–correct**
  An answer must be selected out of five different choices of which only one is correct.
- **Multiple–choice multiple–correct**
  At least one answer must be selected out of five different choices of which more than one can be correct.
- **Any–answer question**, i.e. free format answers
  Up to three text entry fields, which can be completed by the students, are displayed.
- **Hints–and–feedback**
  This type of exercise neither asks questions nor expects answers: it instead provides explanations, gives feedback or introduces a new subject or chapter.

If system developers need additional types of exercises they must write the appropriate programs and add these to the toolset environment. Section 4.2.5.2 explains this subject. [Hall et al., 1996] describe the process of building a resource base of exercises and course documents, which involves collecting, organising, indexing, and linking many types of information, in one phrase:

> [...] reject nothing. It is essential to gather together all the information that might be relevant for a particular subject area. In the initial stages authors should not have any pre–conceived ideas about what is or is not going to be relevant or useful for their particular task [...]. (p106) The important thing is not to focus too closely on that particular task [that is envisaged for the users] during the resource–base building phase. The teacher instead should collect all the resources which either are relevant or might be relevant [...]. (p107)

Adding hypertext links to the generated exercise files is supported by the Tootsie development system. Developers must design a network of links, which represents the intended course flow. Their responsibility is to create a link structure which best suits the different students' learning abilities. Five difficulty levels can be chosen which should provide a more individual learning environment for novices and more experienced students. These levels can alternatively be used to adapt to the motivational learning characteristics of individual students, which can either be failure– or success–driven.

4.2.1.3 Step 3: Resources

The developer must adjust the resource variables of the toolset. These are necessary to adapt the toolset and the tutorial system to the current environment of the computer system on which they are running. The variables do not effect the course or the learning methods and they are not involved in building a resource–base as mentioned before. When the development system is started it will look for the resource files in which information of the toolset and tutorial system environment, e.g. source and target directories, is stored. Appendix A describes each resource variable in detail.

4.2.1.4 Step 4: Templates

The template files define a general layout pattern for each exercise type, but developers can nevertheless adjust the HTML code of the resulting exercises to their needs or add new JavaScript functions. A positive side–effect of templates is that similar code sections, which frequently appear in exercise files, can be recycled: if they are once incorporated into a template, they will then automatically be copied to any new exercises created. In general, this reduces the risk of code errors that may occur if the code sections are manually copied. Whenever authors of tutorial systems use the development system in writing exercises, the template file is filled with the input of the exercise forms and stored as a new exercise in the `Links` directory of the tutorial system. Templates therefore contain special placeholders, called tokens, which are replaced with the information entered by the developer. The tokens are defined in the resource files and must be altered if the token names change. Despite the benefits of templates it is recommended that the interaction between the user and the tutorial system be enhanced by adding individual code to exercises provided this does not interfere with existing JavaScript functions. Consequently, the student's problem solving can be better monitored and analysed: for example, in a complex exercise, in which text entry boxes must be filled out by the learners, the order in which the text is entered can be recorded, and based on this information the tutorial system can react more specifically.

4.2.2 Toolset System Architecture

The toolset is divided into two different groups of files. Firstly, it contains the CGI programs written in C which process the input forms of the user interface. These programs are not used by the tutorial system except for the tools of the cooperative work area and the questionnaire. The developer must copy the compiled CGI programs into a directory which can be accessed by a world–wide web browser. The second group of files is written in HTML, and defines the user interface and the input forms of the toolset. It can be used with any world–wide web browser which is capable of JavaScript.

Tutorial system exercises will be stored in the directories which are defined in the resource files of the toolset. I recommend creating the following subdirectories in the root tree of the tutorial system: `Links` for exercise files, `Applet` for Java applets required by the tutorial system, `Image` for inline images, and `Glossar` for glossary keywords.

4.2.3 Generation

The development system is started by loading the main screen of the user interface with a world–wide web browser. On the left–hand side the table of contents, which lists links to various input forms, is displayed, while on the right–hand side the workspace of the system developer is shown.

4.2.3.1 Step 1: Glossary

Before a glossary keyword can be used in an exercise it must be defined in the glossary form of the Tootsie toolset. Once defined a registered keyword can appear in any introductory text of an exercise, and a suitable hypertext link to the glossary text will automatically be set.

1. **Keyword**
   The developer must enter a keyword which is not yet defined. The keyword must be spelt exactly the same way as it will be used in future exercise texts.
2. **Title**
   Optionally a title can be entered for the glossary keyword.
3. **Explanation**
   The glossary text explains the meaning of the keyword. The developer can use HTML tags, but other

structural information, like newline characters, will be lost. The commands `image(<filename>)` and `applet(<filename>)` include in the text an image or Java applet with the name `filename` provided these are stored in the image or applet subdirectories.

The lower section of the browser window, which is also called frame, shows an overview table of glossary keywords, so the developer can see which keywords are already used. The contents of a glossary file are displayed if the `View` link is clicked.

4.2.3.2 Step 2: Exercises

"Documents should be organised according to whatever (hopefully multiple) structures are required. [...] This is an important phase of the authoring process as this structure provides in many cases the first and primary interface to the information, and as such will tell the user quite a lot about how the author understands the structure of the information, and hence about the subject matter itself ([Hall et al., 1996], pp107/108)". Although this principle is very general, the author of a tutorial system must nevertheless keep it in mind when writing exercises or introductory texts, particularly because CSS which could provide the multiple document structures are not yet available. After selecting an exercise the upper frame of the browser window shows an input form, while the lower frame displays a table with exercises which have already been defined.

1. **Title**
   The filename of a new exercise will be created from the input of the title entry field. As exercises are often part of an exercise section, it is reasonable to store each section in its own subdirectory tree. A directory tree is defined by writing a hierarchy of section names, which start with the topmost section, into the entry field. Section names must be separated by commas, and the hierarchy must end with a unique exercise group[1] name. For directory and file names only the *first* six letters or digits of the section or group names are used. The title line of the new exercise will nevertheless show the complete definition.

2. **Glossary keywords**
   The system developer enters the keywords which must be looked up in the glossary, in form of a comma–separated list. If a keyword is found in the exercise text a hypertext link to the glossary entry will automatically be set.

3. **Help** (not available in hints–and–feedback)
   If system developers want to provide help pages for the students they must enter the following code into the help entry field: {`#<levels>#<text>`}. The help text `<text>` is added to each help page whose help level is mentioned in the preceeding `<levels>`. The character `#` separates the different levels and text definitions, so it is not allowed to appear in `<text>`.

4. **Text**
   Depending on the exercise type the introductory text must be entered and an appropriate question asked. The text can only be structured by using HTML tags. The keywords `image(<filename>)` and `applet(<filename>)` automatically add the object `filename` from the image or applet directories to the exercise.

5. **Answers** (not available in hint–and–feedback)
   Entry requirments vary according to the exercise type: for single–choice and multiple–choice exercises up to five answers can be entered, of which the correct one(s) must be ticked. Any–answer exercises require a question and the correct answer expected from the student. In addition, the keywords `image` and `applet` can be used.

[1] An exercise group is a collection of exercises which have the same title (e.g. they introduce the same knowledge units) but differ in their difficulty levels. An exercise section is a bundle of exercise groups which are part of a common topic or goal.

6. **Difficulty**
   The developer is asked to rate the difficulty level of the current exercise from "trivial" to "difficult". As mentioned before the various difficulty levels of an exercise group can also be applied to provide a learning environment based on the different motivational characteristics of students.

7. **Marked** (not available in hint–and–feedback)
   The developer must decide whether or not an exercise is marked. Only information of marked exercises, e.g. whether it was correctly solved or how often it was tried by the student, is stored in the Cookie variables of the tutorial system on which all the adaption rules and the individual learning support depend. Unmarked exercises however are useful if it is necessary to introduce students to a new topic, which is then completed by a marked terminal page. If an exercise group contains a marked exercise, the group is also "marked", but this does not effect the status of unmarked exercises of the same group.

The `Generate...` button calls the underlying CGI program, and a new exercise file is created. A `Reload` of the lower frame then shows the new exercise entry. The entries consist of a title line, a difficulty level and an exercise type. By clicking on the `View` link the exercise page is displayed in the upper frame.

4.2.3.3 Step 3: Links

A new exercise is integrated into a tutorial system by setting hypertext links to other exercise pages. In order to make this process easier for the system developer, an overview table of existing exercises is displayed: if a `Copy` link is selected and a command button clicked, the necessary information will be copied into the entry fields of the input form. A screenshot of the actual form is shown in figure E.1 on page 101.

1. **Exercise**
   When clicking on `Page` the text entry box will be filled with the data of the exercise, whose `Copy` link was previously selected.

2. **Links**
   The course flow depends on the "link rules", for which JavaScript code must be entered by the system developer for each exercise. The rules will be executed whenever the student selects `Continue...` in an exercise page. In addition, the following commands and JavaScript functions can be used:

   - `link(<filename>);`
     sets a link to the exercise `<filename>`.
   - `link(<group>);`
     sets a link to the exercise group `<group>`. From that group an exercise with the current difficulty level is loaded. If such an exercise does not exist, the tutorial system will try to find the "closest" one. This means that the difference between the current difficulty level and the one of the closest file is minimal.
   - `reset(<filename>);` (only marked exercises)
     resets the flag of the exercise `<filename>` from `successfully done` to `not done yet`. The function `reset` does not work with the current exercise.
   - `isset(<filename>)` (only marked exercises)
     checks whether the exercise `filename` has already been successfully solved. In this case the function will return `true` or `1`.
   - `isset(<group>)`, only marked exercise groups
     checks whether any exercise in the exercise group `<group>` has already been successfully solved. In this case the function will return `true` or `1`.
   - `revisited` (only marked exercises)
     if a student has already solved the current exercise, the variable `revisited` will be set to `true` or `1`.

- `repeated` (only marked exercises)
  if students access an exercise again although they have already solved it, they will be asked in a dialog box whether they want to repeat the same exercise. If they confirm the variable `repeated` will be set to `true` or `1`, `false` or `0` otherwise.
- `correct`
  if a student has correctly solved an exercise, i.e. all the answers correspond to those entered in the input form of the exercise type, the variable `correct` will be set to `true` or `1`, `false` or `0` otherwise.
- `item[<index>]`
  is used as an abbreviation for reading the status of radio buttons, check boxes, or entry fields which the student can tick or fill out in the current exercise. Each of the input objects has an index number which starts with `0` for the topmost object and is increased by one for each object beneath. It is important that for items of single–choice and multiple–choice exercises comparisons like `==`, `!=` etc. are not allowed. In contrast to that, boolean operators, like `&&`, `||` etc., cannot be applied for entry fields of any–answer exercises. If these restrictions are not observed by the system developer, a JavaScript error message will be produced when students access the tutorial system.

3. **Comments**
   Comments or feedback which system developers want to give users on consecutive pages must be defined in the following way: {`#link(<filename>);#<text>`}. `<text>` will only be shown if the current exercise is successfully solved. Again, `<text>` can be structured with HTML tags and the separating character `#` must not appear in `<text>`. Unfortunately `<text>` cannot be source–specific, this means that *any* exercise which is correctly answered and has `<filename>` as its successor will trigger the comment `<text>`.

Clicking on the `Generate...` button will modify the current exercise file by adding the JavaScript code of the link rules.

4.2.3.4 Step 4: Link Reference

If many exercises are linked together, the link structure of a course can become incomprehensible. Therefore, the overview table of all established links assists the developer finding the destinations of source anchors. [Hall et al., 1996] also see the importance of avoiding chaos during the authoring process (p110). Their hypertext system, whose link model is far more sophisticated than HTML, offers two views: first a link index document, which is generated from the linkbase[2] and sorted alphabetically, and second, various documents or linkbases, in which links are sorted according to their purpose (p111). All the source pages are written on the left–hand side of the overview table, while their immediate successors are on the right. By clicking on a destination the overview table will be scrolled to the left–hand side position of the selected exercise. The digits which are written in superscript after the title of an exercise denote the difficulty level of that file. If there is an `x` instead of a number the target page will be selected from the given exercise group according to the user's difficulty level.

4.2.3.5 Step 5: Table of Contents

The table of contents of a tutorial system is not automatically generated. This gives the developer control over what can be directly be accessed by the students, in particular if students have to solve introductory or unmarked exercises before they should proceed. In addition, advanced topics can be hidden, so students do not inadvertently read these beforehand. A good idea is to include documents, which are described by [Hall et al., 1996] as follows:

[2] I.e. a database of links.

> Users need pointers to the important part of the material, which can only be provided in the context of their task. Therefore documents which answer questions for the user such as "What am I doing here as a student?", "what are the objectives and the aims?" and "how am I to go about achieving those?" are not just "nice" but are vital for users to get the most out of their interaction with and use of the information. (pp114/115)

The structure of the table of contents is defined by using HTML tags. However, the reserved space of a table of contents is limited in the current version of the tutorial system Tootsie, so only a narrow list structure is recommended. An exercise group is added by selecting the `Copy` link and clicking on the `Group` button. Instead of the full title for each group only the last portion, generally the title of the file, will appear in the resulting table of contents.

4.2.4 Adaption

Most authors see adaptability as an important feature of tutorial systems for increasing acceptance levels of students. In its most sophisticated form a student model is generated by the system[3], but depending on the used technique lower forms of adaptability can also be applied. [Wenger, 1987] explains three forms of adaptability:

> No intelligent communication can take place without a certain understanding of the recipient. [...] Some systems monitor the student's activity very closely, adapting their actions to the students' responses but never relinquishing control. In mixed–initiative dialogues, the control is shared by the student and the system as they exchange questions and answers. [...] In guided–discovery learning or coached activities, the student is in full control of the activity, and the only way the system can direct the course of action is by modifiying the environment. (pp16/21)

The decision was made to use guided–discovery learning in the Tootsie tutorial system, however with JavaScript the adaptability is reduced to a very basic form. The authors of tutorial systems must therefore responsibly decide what exercises they choose and in which way they present those to the student. [Schulmeister, 1997] has interesting thoughts on the problems which arise in accomplishing adaptability in an intelligent tutoring system. He rightly notes that the functionality of adaption must lead to further differentiation of the learner's parameters because adaptability wants to be "natural", and not coarse or artificial. The increased granularity of the monitored variables however leads to a combinatorial explosion of the diagnosis process, so finally it is only a form of microadaption that intelligent tutorial systems can currently use. [Schulmeister, 1997] unfortunately notes that hermeneutical adaptability, where it is possible for the student to browse in a broad information domain and work with it individually and selectively, is unthinkable for tutorial systems (p201). In many cases only an unlimited information space can promote this level of adaptability.

4.2.4.1 Adaption Variables

As mentioned before Tootsie uses persistent client–state HTTP information, called Cookies, to store a basic student's profile on which the adaption process is based. Whenever the student accesses the tutorial system, the Cookie data will be read and updated on the client's side, i.e. the Cookie file in the student's home directory. In the current version Cookies have a life span of one year, which should be long enough for a course that lasts one semester. The following variables are set by the system:

- `tootsie_cont`
  This Cookie is used by the tutorial system to store which layout was chosen for the table of contents

[3] See page 17.

by the student. Currently there are three settings: "extensive" (`Large TOC`) which displays all entries of the table of contents, "basic" (`Short TOC`) which only shows those entries which are not yet solved, and "annotated" (`I suggest`) which recommends exercises by writing them in bold–faced or italic–faced font.

- `tootsie_dlvl`
  The student's current difficulty level, a numerical value, is stored in this Cookie. The value, ranging from one to five, can be changed either by the student or the tutorial system. In the latter case the system developer must add a JavaScript procedure called `Adaption()` to each template file. The variable is used to load exercises, whose difficulty rating is closest or equal to the current difficulty level.

- `tootsie_hlvl`
  The numerical help level value is used whenever the student selects the help button of an exercise. Based on the help level stored in this Cookie a help file is displayed. If a help file is not defined by the developer, no help button will be shown in the exercise. If a help file is defined for at least one help level, a default text for the remaining levels will be used.

- `tootsie_info`
  The string variable stores for each marked exercise group the difficulty levels of that group that were correctly solved. The information is encoded in alphanumerical form, so each letter represents the status of the whole group. If an exercise is not marked the group value will not be changed.

- `tootsie_name`
  If a new user accesses the tutorial system for the first time he will be asked to enter a name, which is not necessarily his real name. The information is stored in this Cookie and will be displayed whenever the student returns to the start screen of the tutorial system.

- `tootsie_nexe`
  The string variable records how often a single exercise was tried by the student. The information is alphanumerically encoded and saved in a single letter whose binary value will be increased up to a maximum of 25 student accesses. The value will remain unchanged if an exercise is not marked.

- `tootsie_nexr`
  The numerical value counts how many exercises were tried by the student in the current session. The information will be kept until the user clicks on the start button of the welcome screen. It will remain unchanged if an exercise is not marked.

- `tootsie_prev`
  This "true–false" Cookie is used if an exercise is correctly solved by the student. If the system developer enters a comment in the HTML form for defining link rules, the text will be displayed in form of a feedback in the next exercise.

- `tootsie_rsoe`
  The string variable stores for each exercise how many times the exercise was solved by the student until the alphanumerical value — each exercise is represented by a letter — exceeds the maximum of 25 correct solutions. The value will remain unchanged if the current exercise is not marked.

- `tootsie_rsol`
  The numerical value counts how many exercises were solved by the student in the current session. This information will be kept until the user clicks on the start button of the welcome screen. The value remains unchanged if the current exercise is not marked.

- `tootsie_zahl`
  The numerical value counts how many times the welcome screen of the tutorial system was accessed by the student.

System developers can add new Cookie variables, but they should keep the restrictions mentioned in the Cookie spezification by [Netscape Developer, 1997e] in mind. The template files included in the tutorial system, Tootsie, currently offer two procedures to read and write Cookie values: `GetCookie` and `SetCookie`.

4.2.4.2 Adaption Procedure

According to the information which is stored in the Cookie variables the tutorial system can react before the student needs to ask for help or becomes lost. For adapting the course flow to the student, the system developer must add a JavaScript function to the exercise template files. The current version of Tootsie does not have an adaption procedure, but its suggested position in the source code and its name are represented by a deactivated `Adaption()` function call. With the help of the Cookie variables the following questions could be answered:

- Has the student solved a introductory exercise?
- How many exercises have been correctly solved in the current session, and how many has the student tried?
- Is the current difficulty level therefore too high? Or is the help level too low?

The following example shows another approach to adaptability. [Harrer, 1996] describes in his master's thesis how the tutorial system Sypros assumes the student's understanding of a domain by rating the different subtasks of the internal representations of the student's problem solving steps. The moment at which the tutorial system assists the student is controlled by the following parameters:

- Cognitive complexity of a information domain (based on the student's knowledge).
- Motivational disposition of students, which is either more success– or failure–driven.
- Amount of time which has passed since the last assistance.
- Amount of time which has passed since the student had difficulties with the current goal.
- Type of last intervention.
- Difficulty level of the current goal.

In addition, the reactions of the tutorial system, Sypros, are based on "tutoring rules", a catalogue of "if–then–rules" which must be defined by the system developer according to the knowledge domain and didactic principles ([Harrer, 1996], pp85–95).

4.2.5 Flexibility

It is possible to combine the different techniques which can be used for implementing a tutorial system in the world–wide web, to gain the most benefits for interaction between students and system. Incorporating Java applets and plug–ins in HTML documents, on which the tutorial system Tootsie is based, is simple, and so it is possible to offer simulations and interactive graphs to the students. I will however propose other ideas, which describe how Tootsie — in conjunction with its current technique JavaScript — can be modified to introduce more complex teaching schemes.

4.2.5.1 Events

Tutorial systems with a sophisticated student–model, e.g. Sypros, are already in use. They constantly monitor the student's activity and build up an internal representation of the student's knowledge of a given problem. If the student does not follow the presumed path for solving an exercise, the internal representation will respectively be modified. For recording the student's activities, JavaScript code which follows the student's operations must be added to the exercise template files of Tootsie. I suggest using event handlers, which are called whenever user events such as mouse clicks occur. These can be defined for each entry item of an HTML form. For example, if the student changes the value of a text entry box, the event `onChange` will be issued by the browser, on which a JavaScript function can react. The new events

which will be introduced for HTML form objects in the new HTML 4.0 standard, will make possible better understanding of the user's current intentions. There is, however, a major drawback: the size of an exercise file will inevitably increase, and so reading files from the world–wide web will become slower. This can be prevented by using Java applets combined with a database of exercise texts which is queried when user actions occur (like the graph applet of PUSH which is mentioned in Section 3.6.2.1.). If this method is adapted it would, however, be better to implement the tutorial system in the programming language Java rather than JavaScript and Cookies.

4.2.5.2 Exercises

Adding new exercise types to the existing system requires experience in C programming. Nevertheless, the following enumeration describes which steps must be done by a system developer:

1. **Input form**
   An input form for the new exercise type must be defined by the system developer. In this respect, the entry fields "title", from which the filenames will be derived, "exercise text", and "difficulty level", which ranges from one to five, are important. In addition, a radio button is necessary for setting a flag which denotes whether or not an exercise is marked.

2. **Template and resource file**
   The structure of an exercise and the reactions of a tutorial system are defined in a template file. If the new template is similar to the existing ones I recommend using the same tokens which are mentioned in the current resource files. Otherwise a new exercise generator program which is called by an input form must be written.

3. **Generator program**
   The generator program replaces the tokens in the template files for the data, which was entered by the system developer in the input form. All exercises are currently created by different generator programs, which are however based on the *same* source code. For a new type of exercise a new generator program with different source code may be necessary.

4. **Setting links**
   It is necessary to modify the overview tables and input forms which are introduced in Section 4.2.3. In particular, the form for setting links and the link generator program must be adapted to the new exercise type.

A more advanced teaching scheme can be introduced by event handlers and new types of exercises. For complex systems, however, I recommend using a different technique than JavaScript and Cookies. For an intelligent tutoring system storing a permanent student model, providing a sophisticated tutor model, and accessing a profound expert model is necessary. These requirements can only be achieved by using external programs, like Java, knowledge bases, or CGI programs.

4.3 Tootsie Tutorial System

This chapter describes the learning environment of the tutorial system Tootsie. I will explain the general structure of the user interface — its components and their dependencies are shown in figure 4.2 — and give reasons for implementing a cooperative work area. In general, the tutorial system should support the various and individual learning methods of students, but in particular, system developers must concentrate their design issues on the course domain as the user interface should primarily be "context–dependent than student–dependent" (see [Schulmeister, 1997], p41).

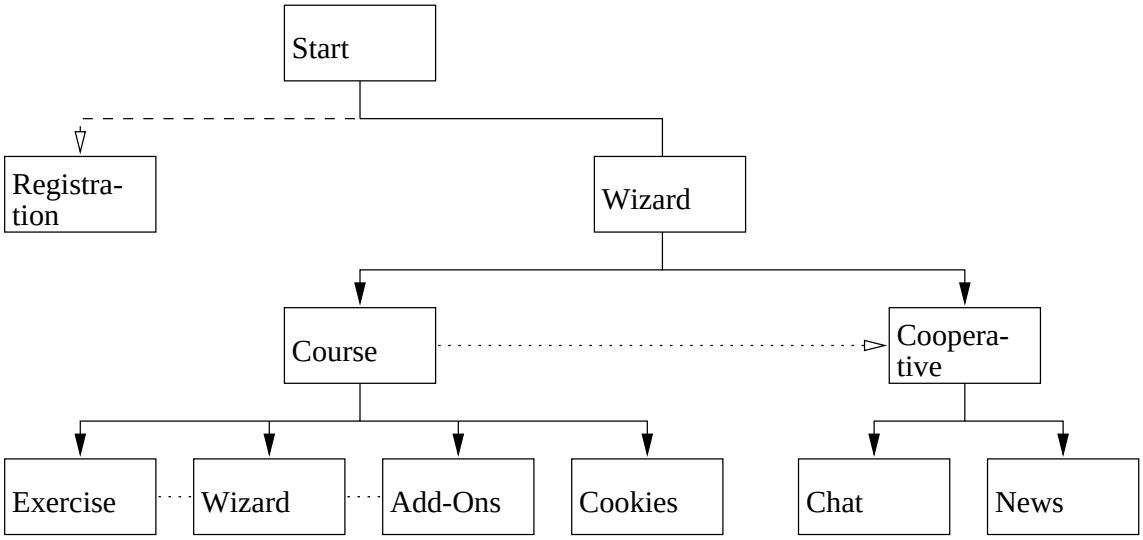


Figure 4.2: After clicking on the `Start` button in the start screen of the tutorial system Tootsie students are either sent to the registration screen (if they are first–time users) or the exercise wizard. From there they can both join the cooperative work environment and the course area of the system. The dotted lines represent easy transfers between the various components of the user interface, for example by clicking on the links in the menu. The latter is only possible from "Course" to "Cooperative" but not vice versa.

4.3.1 User Interface

The tutorial system Tootsie runs in a world–wide web browser window, whose user interface restricts the possibilities for interaction with the student. This is not a severe disadvantage, because most students are already familiar with using a world–wide web browser, so a de facto standard of interaction and navigation in the world–wide web is set. Tutorial system developers should therefore avoid differences between this standard and the interface of the tutorial system itself (see [Espinoza & Höök, 1996]). Although the current version of the HTML is not very flexible[4], designing a well–structured interface is nevertheless very time–consuming, as the usability of the system and the student's learning progress depend on it. Knowledge can be communicated and presented in a more or less intuitive way, so the student's acceptance is primarily influenced by the ease–of–use and the attractiveness of the interface (see [Wenger, 1987], p21). HTML 4.0, and cascading style sheets give the developer the ability to raise that acceptance. Another idea for a hypertext user interface is suggested by [Hall et al., 1996]: they demand the end of the tyranny of the button[5] (p157).

> We encourage our authors to only use buttons when they really need to, but we clearly have a serious re–education problem ahead of us. People who are used to HyperCard or the Web expect links to be indicated by buttons. [...] There is a place for buttons in hypermedia systems [...], but they do not have to be there in order for us to take advantage of the enormous potential of hypertext technology. Drawing an analogy with the way we use books and libraries, we know intuitively when reading a book that we can look up any word or concept we do not understand in a dictionary or encyclopaedia, assuming they are available, or look up any term in the index for a cross–reference, *without the author having to make any indication in the text.* (pp157/158)

The idea is that *any* word in a world–wide web page or hypertext system is a potential link to a definition or explanation of that word. This supports Schulmeister's opinion that hypertext presents a complex learning environment to the students, which allows them to behave in a natural way by browsing the

[4] This will change with HTML 4.0 and its successor, the extensible markup language XML.

[5] In general, "buttons" are highlighted objects which are used to navigate between hypertext documents. For example, in HTML "buttons", i.e. links, are commonly marked by an underlined text.

information space ([Schulmeister, 1997], p271).

4.3.1.1 General

After loading the tutorial system, four coloured sections of the main browser window, called frames, are displayed (as shown in figure E.2 on page 102). Their purposes are:

- **Black**
  The menu frame contains links to the help pages, the cooperative work area, the questionnaire, and the Cookie cutter. There is also a link for quitting the tutorial system by closing the browser window. If the mouse pointer is moved over a link, a more comprehensive description will be displayed in the status line of the browser window.

- **Darkgrey**
  Help documents and explanations of glossary keywords, which were previously selected by the student before, are displayed in "darkgrey" frame. The same frame is used for the add–ons "standard distribution" and "pocket calculator". Both are integrated into the tutorial system to support the student in solving exercises on probability theory and statistics.

- **Lightgrey**
  The frame is reserved for the table of contents of a course, and is currently displayed as a text–based and structured list. A different technique is applied by [Espinoza, 1996]: he uses an interactive graph applet written in Java, which provides an overview of neighbouring and related documents based on the currently selected information unit. The student can simply navigate in the knowledge domain by clicking on the objects of the graph.

- **White**
  The "white" frame is reserved for the student's workspace environment, which consists of exercises, welcome pages, and Cookie cutter information.

The use of frames is often part of controversial discussions on the design of web sites. An advantage is that a frame is an independent section of the browser window, which means that other frames are not influenced, for example by scrolling. A second argument is closely related to the first one: all the frames are constantly visible and, in contrast to windows, they cannot overlap. [Schulmeister, 1997] quotes a 1989 paper by Jonassen which suggests limiting the number of windows that may be opened by the user at any time and expressly forbids overlapping windows in a display (p391). [Schulmeister, 1997] however disagrees: he argues that multiple windows do not detract from the usability of tutorial systems as users get more and more familiar with window systems (p391). In my opinion Schulmeister's arguments are reasonable, but only if the different windows are not constantly needed. This is the case however with the table of contents, the exercise pages and the help texts of the tutorial system Tootsie. A peculiarity of frames is that with Netscape's Navigator 3.x a bookmark can only be set to the start page of a web site, but not to any consecutive documents which are rendered inside a frame. In general, this is a disadvantage, but not in the tutorial system Tootsie, which needs a suitable starting point to reset the system for each session.

4.3.1.2 Menu Items

The menu of the tutorial system is shown in the "black" frame of the browser window. Its contents are:

- **Help, symbol: ?**
  If students require assistance for the different screens of the tutorial system, they can click on the help link in the menu frame. Depending on the displayed document a context–sensitive help page with frequently asked questions is loaded. However, questions regarding specific exercises should only be answered in the "darkgrey" frame.

- **Chat, symbol:** `:)`
  The chat tool, which is part of the cooperative work area, is explained in Section 4.3.2.1.

- **News, symbol:** `!`
  The possibilities of the news group are discussed in Section 4.3.2.2.

- **Email, symbol:** `@`
  After selecting the email link a list of people who are involved in the current course is displayed, so a student can contact the system developer, tutors, or lecturers.

- **Cookie, symbol:** `*`
  The Cookie cutter is described in Section 4.3.1.3.

- **Wizard, symbol:** `A`
  Information on the exercise wizard can be found in Section 4.3.1.4.

- **Feedback, symbol:** `F`
  Students should be able to express their opinion on computer–based learning, so after clicking on the feedback link a questionnaire for comments and bug reports is shown.

- **Quit, symbol:** `X`
  This closes the browser window of the tutorial system.

The menu should be restricted to the most important or frequently used items, and if this is not possible, or submenus are needed, a new menu structure is recommended (for example, with the help of `select` boxes).

4.3.1.3 Cookie Cutter

Cookies are often regarded suspiciously by users, because they are stored in the user's directory and their data is often encoded. Therefore, it is generally difficult to have the user's permission to set a Cookie value. This is the reason why I introduced the Cookie cutter in the tutorial system to allow the students to examine and modify the stored Cookie values. A positive side–effect is that students can also experiment with the tutorial system by trying "what–if"–cases, for example: "what happens if an exercise is marked as solved". More details on the used Cookie variables can be found in Section 4.2.4.1.

- **Overview of exercises**
  The overview table informs students as to whether or not an exercise has been solved. The columns represent the various difficulty levels, so if a student clicks on a check box in the table, the exercise rated with that difficulty level will be marked as solved. The Cookie value is modified by clicking on `Change`.

- **Cookies for adaption**
  This page shows the values of the Cookie variables which store the difficulty level, the help level, the number of exercises tried, and the number of exercises correctly solved. By clicking on the arrow buttons the variables are respectively decreased or increased, and the current value can be displayed by selecting the middle button.

- **Tootsie statistics**
  The two tables tell students how many times they have tried an exercise and how often it was correctly solved. These values cannot be changed by the users.

The Cookie cutter can also be accessed by the system developer for test purposes: by modifying his Cookie values errors and missing links can be found in the system.

4.3.1.4 Exercise Wizard

The exercise wizard is supposed to guide students through the course. It suggests exercises and assists students in planning their curriculum. If users are "lost–in–hyperspace", it will help them to choose a new topic or exercise. Users must normally select the exercise wizard entry in the menu frame, but the developer can also set a link to the wizard in an exercise or terminal page of a course section.

As described in Section 3.2.4 [Weber & Specht, 1997] use two similar adaption techniques, which are called "individual curriculum sequencing" and "link annotation". During their studies however they discovered that the individual guidance only helps learners at the beginning of a course in following an optimal path through the information space. In later sessions the students understood the hierarchical structure of a course, and so most of them did not require further assistance in order to find the best learning path. The exercise wizard will consequently have to face the same problem, but still individual guidance prevents beginners and less skilled students becoming frustrated in the starting phase (see [Weber & Specht, 1997], pp10/11). The following information is offered by the exercise wizard.

- **Difficulty**
  It shows the current student's difficulty level. The remaining information, which is presented by the exercise wizard, is solely based on this difficulty level.
- **Exercises not solved**
  The titles of exercises, which are not yet correctly solved by the student, are displayed.
- **Fuzzy links**
  This section suggests exercises which were once solved, but repeatedly incorrectly answered in later sessions. The wizard assumes that the student has problems with a particular course subject, so it recommends repeating the exercise. In a future version of Tootsie links to additional pages could be included which would give more information on an exercise or try a different approach to the problem.
- **Recommendation**
  The tutorial system suggests whether a student should increase or decrease the difficulty and help level. This advice is based on the number of correctly solved exercises, which is compared with the number of all exercises that have been done during the current session. More specific rules can be implemented by the system developer.

The wizard only processes exercises which are marked *and* listed in the table of contents of the tutorial system. "Hidden" exercises are not supported, because they are primarily used as introductory documents of a chapter instead of terminal pages or work sheets. The system developer should however modify the source code of the exercise wizard in order to assist the student with all the information that fits best to the course. After reading the recommendations the student can choose the next chapter or topic from the table of contents.

4.3.2 Cooperative Work Area

Cooperative learning is especially useful for promoting the acquisition of knowledge because human mental functions and achievments are rooted in social relationships (according to the Russian psychologist L.S. Vygotsky, see [Schaffner et al., 1996], p4). This is, however, not restricted to face–to–face meetings: with computers and computer networks world–wide cooperation is made possible by email communication and computer conference systems. A newsgroup discussion, for example, is often similar to team work: in general help behaviour is increased, hierarchical structures like student–teacher relationships are less important, and the communicating partners have both opportunities to ask *and* answer questions. Therefore, goals are faster and better achieved than in computer–assisted, but competitive and individualistic learning environments (see [Schulmeister, 1997], pp283/284).

The students' individual learning characteristics greatly influence the success of cooperation. For skilled students the learning method is less important, but less skilled learners mostly benefit from the cooperative method. The advantage is that problems can be discussed with other students, thus errors are quickly detected and more suitable algorithms for solving a particular problem are remembered (see [Schulmeister, 1997], p284). However, I must note that most studies which compared cooperative and non–cooperative learning methods have not found any significant differences (see [Schulmeister, 1997], p284). I was nevertheless convinced of the benefits which cooperative learning could offer, so I introduced the following two tools in the cooperative work area. In particular, benchmark lessons, which are introduced in Section 4.3.2.2, offer a new form of discussion which increases the student's knowledge better than newsgroups and addresses more experienced learners.

4.3.2.1 Chat

In general, a "chat" is a program or a world–wide web site where different users, who often do not know each other, meet in order to discuss previously agreed topics. The "chatters" enter sentences with the help of their keyboards, submit the text to the chat group and read the answers on the computer screen, which is frequently updated, so all participants are able to follow the discussion. According to [Sackl, 1997] this form of communication is called "synchronous", because messages immediately reach the communicating partners and can be answered at once if the partners are present at the same time (p47). Newsgroups, which are described in the following Section 4.3.2.2, represent an "asynchronous" communication tool and need the ability to store messages which are sent by the various partners[6] on disk, because members of a newsgroup rarely take part in a discussion at the same time. Therefore, they must have access to all the messages which have been issued after their last visit. A message can only be read by the participants of a discussion after it has been posted. However, compared to the few seconds in synchronous communication, the delay between posting a message, and the message becoming available to the other participating members is not insignificant.

> Online asynchronous discussion lacks the real–time (synchronous) feedback of a live discussion. However, it does allow students to compose their thoughts carefully, develop written communication skills, work away from school and have time to use references when considering their responses. At the same time, we acknowledge that rewarding discussions must be active and timely ([Schaffner et al., 1996], p16).

In the world–wide web chats belong to the most frequently visited sites, so their use is already quite common and accepted. By adding a basic chat program I intend to give the students the possibility of talking with each other about problems, solutions, and exercises. At least once a day a human tutor or lecturer should be present, so he can also be contacted with the help of the chat tool. In the internet the communication is often not moderated which means that people do not know with whom they are talking, so this may be an advantage for students who feel insecure when talking to others face–to–face, for example in a lecture etc. In a chat these students can participate more freely, and with the Tootsie chat program they can also remain anonymous. The chat tool is selected by clicking on the link in the menu frame of the tutorial system. Two new frames are displayed: the lower one contains the discussion text while in the upper frame messages can be entered. At the beginning students are also asked for their names[7], which are then fixed for the current chat session. Messages are immediately sent to the discussion, when the submit button is clicked. From that moment on the text frame is updated every ten seconds, and students are able to participate in the discussion at any time. Their contributions are shown with their names, the date, and the time of the posted message. In general, it is possible to introduce different chats for various topics or groups of learners in order to offer an environment to the students which is better based on their requirements.

[6] This process is also called "posted".
[7] Called "avatar".

4.3.2.2 News

The implementation of the news tool was inspired by the benchmark lessons which are mentioned in the paper by [Schaffner et al., 1996]. In general, the current tool can be used to simulate newsgroups commonly known from the usenet, however the benefits of benchmark lessons persuaded me to introduce the same ideas for the tutorial system, Tootsie. Benchmark lessons are based on "facets" which [Schaffner et al., 1996] use as the building blocks of their tutorial system, DIANA. In general, facets are described as pieces of knowledge that compose a person's understanding. They represent ideas which students have obtained from a certain domain, and it is the task of a human or computer–based tutor to identify pre–existing and context specific facets in order to build instruction upon them. A key factor of facets is that they can be recognized whenever the student uses them, for example in writing or speaking. Therefore, a tutorial system can collect the various facets, and upon that decide which advice must be given to enhance and extend the student's skills and expertise. The development of a facet database and the uses of facets in learning are thoroughly explained in [Schaffner et al., 1996].

A benchmark lesson is a full–class discussion moderated by an instructor[8] and designed to provoke group discussion of the facets held by the different students (p10). [Schaffner et al., 1996] believe that benchmark lessons in conjunction with lectures and case–studies stimulate thought and promote students' social interaction as learners must act together and occasionally change their attitudes and ideas (p10). In a benchmark lesson a problem is presented to the students by the moderator or lecturer, and from that moment on the participants of the discussion are asked to solve the task. [Schaffner et al., 1996] recommend structuring the postings of a benchmark lesson, so new facets can be created, challenged and reflected by the students. A sample benchmark lesson is described in [Schaffner et al., 1996, Chapter 3.4], which I summarise here:

> Our solution balancing the merits of asynchronous and synchronous discussion imposes a fairly rigid structure and schedule on the discussion. Our virtual benchmark lesson has four primary parts: (1) initial response and justification, (2) critique, (3) discussion and (4) reflection [...]. Social loafing may occur when individuals do not feel that their participation is necessary for the group to function fully. Rogelberg et al. (1992) introduced the "step–ladder technique"; a group discussion mechanism that requires every group member to submit their ideas before any thorough discussion or conclusion [...]. Once all initial responses and justifications have been posted, students read each other's posts and are required to critique at least one other post by either arguing in favor or against it while providing support and examples for their position [...]. We wrap up the discussion with reflection. Again, temporarily blinded to other responses, each student makes a final contribution to their group's collection of posts by contributing a short summary of what was learned in the benchmark lesson.

The advantages and disadvantages of benchmark lessons are discussed by the authors:

> On–line collaboration gives each student the opportunity to voice their opinions and participate [...]. In the virtual environment everyone has a chance and the environment feels less confrontational [...]. High–ability students may benefit from explaining ideas to low–ability students, gaining the intellectual benefit from teaching [...]. Low–ability students may benefit from having peers explain concepts in terms closer to their own understanding [...]. Virtual benchmarks are valuable to the instructor. Because discussion proceeds openly without judgment, teachers are able to get a deeper idea of what is and is not understood by the class as a whole [...]. With every student participating in many ways and at many levels, it is very difficult for an instructor to give each response the careful and thorough attention it deserves [...]. Another problem of the virtual benchmark is that there is no way to ensure that each student reads all the other postings in the group.

[8] These intentions are in contrast to the more open usenet newsgroups.

As mentioned above the benchmark lessons encompass many benefits which asynchronous communication offers in the work with discussion boards. The strict structure of benchmark lessons is especially important if bigger groups of students are participating. [Schaffner et al., 1996] recommend restricting the number of users to 8 or 10 by splitting up a discussion board into smaller ones. The experiments carried out by the authors showed how well benchmark lessons were received by the students. "Researches have shown that discussion methods keep students active and involved with their learning. Ideally benchmark lessons follow diagnosis and are chosen based on the prevalence and severity of novice facets in a classroom; however, benchmark lessons (in–class or on–line) are likely to be beneficial to any audience, even when divorced from diagnosis (p32)." In my opinion it may be precarious to abolish diagnosis that tries to find out the students' novice facets on which the design of benchmark lessons should be based. [Schaffner et al., 1996] earlier say in their paper that effective instruction should identify students' pre–existing and context specific pieces of knowledge, build upon them and weave them into a coherent whole (p5). Therefore a developer of a tutorial system should include a method which makes diagnosing facets at the very beginning of the design process possible. Nevertheless, I emphasize that some of the results, which are mentioned in [Schaffner et al., 1996], are not universally applicable: the tested user groups may not be representative, new teaching methods are often positivly considered at the beginning but this effect may wear off, and the results may be influenced by the Hawthorne effect.

4.3.3 System Evaluation

Restrictions that arise from using JavaScript and Cookies, are explained in Section 3.5.4, so in this section I present the results of a beta–test, which was done with a small tutorial system at Technische Universität München in April 1997. The first trial version was not aimed at testing the educational abilities, but examining the technical availability and usability of the system. I announced the test in one of the computer science newsgroups at Technische Universität München, and eight students anonymously participated in a course consisting of four basic questions on Bayesian laws which were taken from the DIANA tutorial by [Schaffner et al., 1996]. The results are not scientifically relevant, because the user group was very small and does not represent the student body of the university. In addition, the small number of exercises and the missing difficulty levels prevent an educational analysis. The test offered one exercise in form of a benchmark lesson, but the students did not participate. For the implementation I used the German version of the tutorial system Tootsie, which did not have an entry questionnaire for assigning the student to a user group that is distinguished by the difficulty and help levels. The system was rated by answering an online questionnaire which could be filled out by the students at any time. The Cookie values of each student could optionally be included, so I was able to see how often and how successfully a student had worked on the course. The results were promising, but some previously unknown problems arose. These are explained and solutions are suggested:

- "HTML and JavaScript code was displayed in the document window."
  This problem had not shown up in my previous tests, and I do not exactly know what happened. I think the reason for this problem is based on the method by which a framed document is loaded by the world–wide web browser: occasionally not just one frame must be updated, but two or more at the same time. Presumably this is not properly done by the browser, because selecting `reload frame` in the menu correctly works for each individual frame. I recommend restricting the use of frames to situations where not more than one frame must be updated.

- "It was difficult to return to the start screen of the system."
  This is intended, because the start screen resets all the Cookie values, and therefore it should only be accessed at the beginning of a session. The problem can nevertheless be fixed by putting a link to the start screen in the table of contents.

- "The dialog box which asks whether or not the student would like repeat the exercise, was displayed many times. I recommend using a neutral screen instead."
  I must emphasize that the course only consisted of four exercises, so this dialog box frequently appeared. In a larger system this will rarely be the case. I can nevertheless offer a different solution: the developer must introduce a new Cookie variable that records, for each exercise, whether or not the student has answered the dialog box with "no". Whenever a link to a next page is returned by

a JavaScript function, a new procedure must be called which checks the Cookie variable, and in case of a "no" entry the exercise wizard is loaded.

- "The used font is too small."
  A problem of world–wide web browsers is that their settings or ways of displaying a web page are very different. Before cascading style sheets are wide–spread, world–wide web users must be asked to select a new font in their preferences.

- "The start screen is not well organized, and buttons or links are not explained."
  The idea of separating the start screen from the other pages — currently the table of contents, the menu, and the help frame are immediately displayed after the system was loaded — will be adopted in future releases. For explanations of buttons and links, the status line of the browser can be used, so the system developer must modify the C source and template files of the Tootsie Development System.

- "Tootsie did not work with Microsoft's Internet Explorer 3.02."
  For Microsoft's Internet Explorer 3.x the use of frames is not recommended, because documents cannot be loaded into a frame different from the one where the update request was issued. The system developer must therefore consider whether or not it is possible to work with a single document window.

The remaining questions of the questionnaire were answered by the students in the marking scheme of the university: the lower the average mark is, the better the topic was rated. The results are presented in table 4.1.

Table 4.1: Results of the Tootsie Tutorial System Evaluation in April 1997

	Student								
Topic	A	B	C	D	E	F	G	H	**Avg**
Fun in Working with Tootsie	2	3	3	2	2	3	4	n/a	**2.7**
Own Educational Progress	5	5	2	2	n/a	5	5	n/a	**4.0**
Usability of Tootsie	2	3	2	2	1	4	4	n/a	**2.6**
Benefit of Group Work Area	n/a	2	2	3	5	n/a	n/a	2	**2.8**

Chapter 5

Conclusion and Outlook

This thesis introduced the fundamentals of tutorial systems by discussing the psychological background, the various types of educational software, and the prototypal implementation of a sample system. Its primary goal was the comparison and the presentation of techniques which combine tutorial systems and the world–wide web to a learning environment that is both universally accessible and individually adaptable. Based on the previous chapters I can conclude that the future prospects of distance learning are good: plug–ins and Java applets offer real interaction between student and system, external programs are integrated with the help of CGI (or the CL–HTTP server respectively), and HTML 4.0 and JavaScript provide new ways of structuring documents and monitoring the students' activities in formerly static HTML forms. The advantages of a world–wide web based course are the availability and the independence of the system. Students can access tutorial systems without general restrictions of time or location, so a learning environment is created which is focused on the learners' needs, and not on organisational preliminaries. In this respect the underlying world–wide web technology is less important than the pedagogical concepts. Most of the resources reserved for the system design should therefore be invested in establishing an exploratory and motivational environment which challenges the users and supports them according to the integrated didactic principles.

This thesis was also aimed to provide an overview of the current research of tutorial systems. The combination of the subjects Psychology and Computer Science is manifested by the proposed implementation for knowledge representation, feedback, adaption, and the structural organisation of tutorial system components. Future software will strengthen these ties: as research of artificial intelligence and computational linguistics[1] proceeds previous modelling problems, like true understanding of the students' actions, may be solved. Practical solutions for the "use of the button", the organisation of course documents, adaption, and a cooperative work environment can be found in Sections 4.2 and 4.3. Some of these proposals have already successfully been applied to various tutorial systems, so their integration in existing or new educational programs should be considered. In the future distance learning will play an essential role in adult training as the various research projects, like Lecture 2000[2] or Deutsche Telekom's T–Mart[3], show. However, tutorial systems cannot fully replace human communication: firstly, social interaction is often essential for learners (see Section 4.3.2), and secondly, current tutor and student models are still inadequate to react specifically on the individual problems which hinder successful learning. In its current form distance learning is therefore a supplement to traditional training rather than an alternative.

Ideas which use the different world–wide web technologies to implement a tutorial system are presented in Section 3.7. However, with their help it is hardly possible to compensate deficiencies of the user interface of the underlying browser application. For a world–wide web based tutorial system two features, which current browsers do not provide, would be extremely useful:

- **Magic marker**
  If students read documents which are printed on paper they often mark passages, which are important for them, with a magic marker and write annotations beside the text segments. This technique could easily be implemented in world–wide web browsers: text is highlighted by the users and annotated with the help of an external text editor. Whenever the users return to the same page,

[1] For example, see http://www.cis.uni–muenchen.de/.
[2] see http://wwwschlichter.informatik.tu–muenchen.de/proj/lecture/.
[3] see http://www.t–mart.com/.

which is uniquely characterised by its URL, the text segments are automatically highlighted and the entered information is displayed if the students click on the marked passages. The annotations and the locations of highlighted text could be stored on the users' hard disk.

- **Storing of contents**
  Currently, if world–wide web users save HTML documents on their hard disks, the embedded objects, like images, will unfortunately not be downloaded. The possibility of storing the contents of a web page automatically could be implemented by transfering the objects to a local directory and by replacing the links of the remote objects, which are inside the HTML document, with links to the new file names.

The prototypal implementation of Tootsie provided an insight into the possibilites and restrictions of tutorial systems on the world–wide web and JavaScript in particular. I presented arguments for and against JavaScript, introduced a development system for creating exercises as easily as possible, and examined various enhancements, like the exercise wizard and the cooperative work environment, of the traditional drill–and–practice approach of the tutorial system. An evaluation, which was held in April 1997, showed how students accepted the design of my computer–based training program. The following enumeration discusses limitations of my solution and presents implementational alternatives:

- For creating exercises and HTML documents the development system was introduced to prevent superficial errors being made by system developers, which for example frequently occur when code sections are manually copied. The exercise templates however, which define a general layout pattern for an exercise, restrict the flexibility of the final system by excluding more individual course documents. This problem can either be solved with the integration of new templates or with manual modifications of the resulting HTML files.

- The cooperative work environment allows students to interact with each other, for example by discussing a benchmark lesson. The current tools are not fully operational: the programs do not support locking of files, so whenever two processes simultaneously access the same file an error message will be returned. The relevant components can easily be replaced, so for example a "chat" program which is based on the programming language Java could be used instead.

- The system code is solely implemented in JavaScript which allows the quick integration of individual system functions. This suits the indended use of the system as a drill–and–practice program with basic extensions for student adaption. For more advanced training software however, a JavaScript implementation would be better replaced by Java or (partly!) CGI programs. These allow more extensive student monitoring, bigger systems, and the "separation of concerns".

- The user interface of a tutorial system is supposed to relieve the work–load of the students' working memory, so a well structured and intuitive design is recommended. The current implementation is only used for test purposes, and should therefore be changed.

This thesis aimed to provide an overview of technologies which are available for a broad user group. Therefore, it does not describe methods which are rarely used (like the virtual reality modelling language VRML), still in the making (for example, the extensible markup language XML), or platform–specific (like Microsoft's ActiveX). These techniques should nevertheless be considered if a tutorial system is designed as they often provide possibilities which other technologies do not offer.

Appendix

Appendix A

Resource Variables

The following resource variables are needed to localize the Tootsie development system. For example, they specify in which directories files are stored, which tokens introduce important code sections, and which error messages are returned to the user. The letters C, G, L, M, S, T, and V, which are described in Appendix B.1.1, represent the various resource files .rsc, in which the variables are defined. These files are used by the CGI programs of the Tootsie development system.

Table A.1: Resource Variables of the Tootsie Development System.

Resource Variable	Usage	Comments
APPLET_DIRECTORY	-G-MSTV	The world–wide web directory, i.e. URL path, which contains all the Java applets for the tutorial system.
BASE	---MSTV	Token in the template file that is replaced with the contents of BASE_HREF.
BASE_HREF	CGLMSTV	The world–wide web root directory, i.e. URL path, where all the exercises of the tutorial system are stored.
CONTENT_ENDMARKER	C-LMSTV	String or character which ends the overview list of exercise sections.
CONTENT_FILE	C-LMSTV	Development system file in which information on the exercise sections is stored.
CONTENT_MARKER	C-LMSTV	String which starts the overview list of exercise sections in the file CONTENT_FILE.
COOKIE	---MS-V	A token which is replaced with the position of the Cookie character in the Cookie string, that is used for the current exercise.
DEFAULT_HELP_TEXT	---MS-V	The default help text which is used for a help file if no other text is defined by the author.
DEFAULT_HELP_TITLE	---MS-V	Default title of a help file.
ENTRY_SYNTAX_ERROR	C-L----	Error message which is displayed on screen if wrong data has been entered into the current form.
EXERCISE_TYPE	---MSTV	A two–letter code which denotes the type of an exercise. It is used by various programs and files of the development system.
EXERCISE_TYPE_TEXT	---MSTV	A description for the two–letter code EXERCISE_TYPE which is required in the overview table of exercises.
EXERCISE_WIZARD_FILE	C------	The location and name of the exercise wizard file.
FILE_END	C-LMSTV	Text which is returned to the author if a program of the development system has successfully processed the form input.

...

...

Resource Variable	Usage	Comments
GLOBAL_ENDMARKER	--L----	String or character that ends the section of an exercise file, in which the global variables are defined.
GLOBAL_MARKER	--L----	String or character that starts the section of an exercise file, in which the global variables are defined.
GLOSSAR_DIRECTORY	-G-----	The directory in which all the glossary definitions are stored.
GLOSSAR_END	-G-----	Text which is returned to the author if a glossary file has been successfully written on disk.
GLOSSAR_INDEX_ENDMARKER	---MSTV	String or character which ends the overview list of glossary definitions in the file GLOSSAR_INDEX_FILE.
GLOSSAR_INDEX_FILE	-G-MSTV	File name, including path, in which the overview list of glossary definitions is stored.
GLOSSAR_INDEX_MARKER	---MSTV	String or character which marks the start of the overview list of glossary definitions.
GLOSSAR_SEPARATOR	---MSTV	Glossary keywords which are used in an exercise text must be specifically entered by the author, so links to the glossary files are automatically set by the development system. Consequently, GLOSSAR_SEPARATOR defines the character which separates the different keywords.
GLOSSAR_SYNTAX_ERROR	-G-----	Error message which is displayed if a syntax error has been found in a glossary file.
GLOSSAR_TEMPLATE	-G-----	File name, including path, of the glossary template file.
HELP	---MS-V	Token which is replaced by the development system with the help file path for the current exercise.
HELP_DECR	---MS-V	Token which is replaced with the filename of a help file that has a lower help level than the current one.
HELP_FILENAME	---MS-V	This resource variable is currently not used. In future releases it will specify the filename of a help file, which is by default assigned to a help level whenever a help text is not entered by the system developer for this particular level. To activate the resource variable the file `exfunc.c` must be edited.
HELP_INCR	---MS-V	Token which is replaced with the filename of a help file that has a higher help level than the current one.
HELP_SYNTAX_ERROR	---MS-V	Error message which is displayed whenever the input of the help entry box is incorrect.
HELP_TEMPLATE	---MS-V	Path and filename of the template file for help pages.
HELP_TEXT	---MS-V	Token which is replaced with the help text by the development system.
HELP_TITLE	---MS-V	Token which denotes the position of the title string in a help file.
HINT_ENDMARKER	--L----	String or character that ends the section which contains the feedback text. This text is displayed to the user whenever the previous exercise has been successfully solved.
HINT_MARKER	--L----	String or character that starts the section which contains the feedback text.

...

...

Resource Variable	Usage	Comments
IMAGE_DIRECTORY	-G-MSTV	The world–wide web directory, i.e. URL path, that contains all the images which are used by the tutorial system.
INDEX_FILE	--L----	The location and filename of the overview list, in which all the relevent information on linked tutorial systems exercises is stored.
INDEX_ENDMARKER	-GLMSTV	Generally it defines the end delimiter for the overview list of created exercises. For the link program, however, it symbolises the end of the section, which contains the link information.
INDEX_MARKER	-GLMSTV	Generally it defines the start delimiter for the overview list of created exercises. For the link program, however, it symbolises the start of the section, which contains the link information.
INFO_ENDMARKER	--L----	It marks the end of the code section in an exercise file where links to succeeding exercises are stored.
INFO_MARKER	--L----	It marks the start of the code section in an exercise file where links to succeeding exercises are stored.
LINKS_DIRECTORY	--LMSTV	It defines the root directory in which the created exercises will be stored (the path must be similar to the world–wide web directory BASE_HREF).
LINKS_INDEX_ENDMARKER	--L----	It marks the end of the overview table for created exercises.
LINKS_INDEX_FIlE	--LMSTV	Location and filename, in which the overview table for created exercises is stored.
LINKS_INDEX_MARKER	--L----	It marks the start of the overview table for created exercises.
LINKS_TEMPLATE	---MSTV	Location and filename of the exercise template.
LOGIC_END	--L----	This marker is placed into the exercise template but required by the linking program. It defines the end of code section, which is executed when the user selects Continue....
LOGIC_START	--L----	It defines the starting point of the code section, which is executed when the user selects Continue....
MULTIPLE_CHOICE	--L----	The user interface of the development system does not use the common abbreviation mc for multiple–choice questions, but shows a more intuitive expression instead. Therefore, for the link program the string MULTIPLE_CHOICE specifies which term stands for mc.
NO_CONTENT_ENTRY	--L----	Error message which will be displayed if the list of exercise sections is not found or if an erroneous entry is read.
NO_CONTENT_FILE	C-LMSTV	Error message which will be displayed if the file with list of exercise sections is not found.
NO_CONTENT_MARKER	C------	Error message which will be displayed if the start delimiter for the list of exercise sections is not found.
NO_ENDMARKER	-G-MSTV	Error message which will be displayed if the end delimiter of a list is not found.

...

...

Resource Variable	Usage	Comments
NO_EXERCISE	--LMSTV	Error message which will be displayed if the exercise file is not written on disk.
NO_EXERCISE_WIZARD_FILE	C------	Error message which will be displayed if the file EXERCISE_WIZARD_FILE cannot be opened by the system.
NO_GLOSSAR	-G-----	Error message which will be displayed if the glossary file is not written on disk.
NO_GLOSSAR_INDEX	---MSTV	Error message which will be displayed if the file containing the overview list of glossary keywords is not found.
NO_GLOSSAR_LINE	---MSTV	Error message which will be displayed if an error in the file with the overview list of glossary keywords is found.
NO_GLOSSAR_MARKER	---MSTV	Error message which will be displayed if a delimiter for the overview list of glossary keywords is not found.
NO_HELP_FILE	---MS-V	Error message which will be displayed if a help file is not written on disk.
NO_HELP_LEVEL	---MS-V	Error message which will be displayed if the author does not specify a help level in the entry field for help texts.
NO_HELP_TEMPLATE	---MS-V	Error message which will be displayed if the template file for help pages is not found or cannot be opened.
NO_INDEX	-GLMSTV	General error message which will be displayed if an overview file cannot be found.
NO_LINKS_INDEX	--L----	Error message which will be displayed if the file containing the overview list of the created exercises is not found.
NO_MARKER	-G-MSTV	Error message which will be displayed if a delimiter of a overview list is not found.
NO_TEMPLATE	-G-MSTV	Error message which will be displayed if the file containing the exercise template is not found.
NO_TOOTSIE_CONTENT	C------	Error message which will be displayed if the content file cannot be written on disk.
SINGLE_CHOICE	--L----	Similar to MULTIPLE_CHOICE, but used for single–choice questions instead.
SOLUTION	---MS-V	This marker, which is used in an exercise template, defines the start of the code section where the solutions to the current questions are stored.
SUBTEXT	---MS-V	This marker, which is used in an exercise template, defines the start of the code section which is responsible to display the exercise questions.
SUBTEXT_ENDING	---MS-V	Source code which must follow after the code section which is responsible to display the exercise questions.
SUBTEXT_HEADER	---MS-V	Source code which must precede the code section where the exercise questions are written.
SUBTEXT_NUMBER	---MS-V	Code portion which displays a numberic counter for questions.
SUBTEXT_SOLUTION	---MS-V	Code portion which encapsulates the form element of a question, which can be selected by the student.

...

...

Resource Variable	Usage	Comments
SUBTEXT_SOLUTION_TYPE	---MS-V	Code portion which displays the form element, which can be selected by the student.
SUBTEXT_TEXT	---MS-V	Code portion which encapsulates the text that is asked in a question.
TEXT	-G-MSTV	Token which is replaced with the introductory text of an exercise or glossary entry by the development system.
TITLE_SEPARATOR	---MSTV	Like GLOSSAR_SEPARATOR it specifies a character which separates each element of the title string. The last part is used to generate the filename of an exercise, while the others are needed to create suitable subdirectories.
TITLE	-G-MSTV	Token which is replaced with the title of an exercise or glossary entry.
TOOTSIE_CONTENT	C------	Filename, including path, of the table of contents of the tutorial system.
TOOTSIE_CONTENT_ENDMARKER	C------	String or character which ends the code section that defines the structure of the table of contents.
TOOTSIE_CONTENT_MARKER	C------	String or character which starts the code section that defines the structure of the table of contents.
TOOTSIE_HOME	CGLMSTV	Home directory of the Tootsie development and tutorial system.
TOOTSIE_TEXT_ENDMARKER	C------	String or character which ends the code section that is responsible for displaying the table of contents.
TOOTSIE_TEXT_MARKER	C------	String or character which starts the code section that is responsible for displaying the table of contents.
VARIOUS_ANSWERS	--L----	Similar to MULTIPLE_CHOICE, but used for various–answer questions instead.
WORD	-G-----	Token which is replaced with the glossary keyword.

.

Appendix B

Tutorial System Source Files

Tootsie is a suite of C and HTML files. The program files, which are required to create exercises, must be compiled with an ANSI C compiler, before their executables can be copied to a directory of the world–wide web server which is reserved for CGI programs. The HTML files either define the user interface of the development system or the work environment of the tutorial system.

B.1 Tootsie Development System

B.1.1 Common Gateway Interface Source Files

The abbreviation **RF** stands for **r**esource **f**ile, so each letter which is used in the **RF** field represents a file that defines resource variables of the Tootsie development system. These variables are described in Appendix A. If the **RF** column contains one or more letters, the specified resource files will be read by the executables created from the compiled program code.

Table B.1: CGI Source Files of the Tootsie Development System.

Source File	RF	Comments
`chat.c`		The chat program is part of the cooperative work tools. It uses two text files, which are defined in the header of the source code, to generate the output of the chat. Whenever a participant says something, i.e. output must be written, the changes are first made in a backup file, which will replace the standard output file then.
`content.c`	`C`	This program file is responsible to write the table of contents for a tutorial system as it is specified by the author. Localized data is read from the resource file `tdevco.rsc` (`C`).
`exfunc.c`		It contains the subroutines which are called by `exmain.c` whenever an exercise file is generated.
`exfunc.h`		It defines the object structures and procedure signatures that are required by `exmain.c`.
`exmain.c`	`M` `S` `T` `V`	The main program, which generates exercise files, calls the subroutines of `exfunc.c` in order to parse the developer's entries in the exercise forms. It also defines the names of the resource variables and reserves memory space for them. Localized data is read from the resource files `tdevmu.rsc` (`M`), `tdevsi.rsc` (`S`), `tdevti.rsc` (`T`) and `tdevva.rsc` (`V`).
`exmain.h`		The header file of `exmain.c` contains preprocessor data and provides access to the internal representations of the resource variables.

...

...

Source File	RF	Comments
glossar.c	G	The program is responsible to write a glossary file according to the author's input in the glossary definition form. Localized data is read from the resource file tdevgl.rsc (G).
linking.c	L	The program connects the different exercise files with hypertext links as specified in the input fields of its world–wide web form. Localized data is read from the resource file tdevli.rsc (L).
mailme.c		The program enables users to give feedback in form of a questionaire during the beta–test of the system.
Makefile		A standard makefile for compiling the executables from the source code.
news.c		As part of the cooperative work tools the program implements a basic news group for the course. Localized data is read from the resource file abnews.rsc.
*.rsc		Resource files which set the environment variables of the Tootsie development system. These variables are used to localize the Tootsie development and tutorial system.

.

B.1.2 User Interface and System Files

Table B.2: User Interface and System Files of the Tootsie Development System.

System File	Comments
tdev10.htm	The file is the start page of the Tootsie development system. Two independent sections of the browser window, which are called frames, are build up and the files tdev11.htm and tdev12.htm are loaded into these.
tdev11.htm	It stores the table of contents of the development system. In order to create exercises etc. for a tutorial system the developer should follow each link from top to bottom.
tdev12.htm	This page gives information about the current version of the development system.
tdev20.htm	An overview page with detailed instructions on how to create exercises for the tutorial system. It also contains further links to the individual input forms of the different exercise types.
tdev21.htm to tdev24.htm	Basically a single HTML file for displaying two browser frames. The first one is used for the various input forms, whereas the other one shows the overview table of created exercises, which are stored in tdevex.htm.
tdev25.htm to tdev28.htm	These contain the input forms for the different exercise types.
tdev30.htm	Two frames are defined by tdev30.htm. One shows an input form for glossary keywords, the other one the overview table of existing glossary entries, which is stored in tdevgl.htm.
tdev31.htm	It displays an input form for glossary keywords.
tdev40.htm	Two frames are defined in order to display the files tdev41.htm and tdevex.htm.
tdev41.htm	This input form must be used to connect exercises with hypertext links.
tdev50.htm	Firstly the overview table of existing hypertext links, which is stored in file tdevlk.htm, is displayed. Secondly, the file tdevex.htm is loaded to enable system developers to access exercises and check their appearnace by selecting the View link.

...

...

System file	Comments
tdev60.htm	Two frames are defined in order to display the two files tdev61.htm and tdevsc.htm that are necessary to generate the table of contents for a tutorial system. The first frame contains the input form, whereas the second one shows the overview table of existing exercise sections.
tdev61.htm	The input form for creating a table of contents.
tdev70.htm	As the tutorial system is using Cookies to store information on the student's progress, it is necessary to give the students access to the Cookie values. The file tdev70.htm shows an overview of all possibilities that exist in order to modify Cookies.
tdev71.htm to tdev73.htm	These HTML pages enable students to change Cookie data.
tdev80.htm	Generally, if the developer clicks on help, a JavaScript procedure is called to find out, what input form or HTML page is currently displayed in the world–wide web browser. However, if this does not work, i.e. the help link is case–insensitive, the file tdev80.htm is loaded instead.
tdev81.htm to tdev89.htm	These files contain the help pages for the development system. If the help link is case–sensitive, one of these files will be shown immediately, otherwise the developer must select them from tdev80.htm.
tdevex.htm	It stores the overview table of created exercises whose entries consist of a title, HTML filename, difficulty level, and exercise type.
tdevgl.htm	It stores the overview table of glossary terms whose entries consist of a glossary keyword, HTML filename, and title.
tdevlk.htm	It stores the overview table of existing hypertext links whose entries consist of the title of the source file and the titles of the succeeding exercises.
tdevsc.htm	It stores the overview table of exercise sections whose entries consist of a title for each section, a path portion which is relative to the root directory, the difficulty levels, and the Cookie index which is either increased for each marked section or 0 otherwise.
tdev?t.htm	Files with the pattern tdev?t.htm are generally used for exercise templates. Templates can be edited by the developer and represent the layout of an exercise. Whenever the developer creates an exercise by filling out the appropriate HTML form, the tokens of the template are replaced by the contents of that form. For this operation the token names must be defined in the resource files.

.

B.2 Tootsie Tutorial System

B.2.1 User Interface and Work Files

Table B.3: User Interface and Work Files of the Tootsie Tutorial System.

Work File	Comments
abchat?.htm	These files are used by the chat program, a tool that is designed to offer a synchronous cooperative work environment.
abemail.htm	It is recommended to list the email addresses of all the people who are currently involved in the design of a course in this file.
abfirst.htm	The file abfirst.htm is an on–line assistant, called exercise wizard, which suggests or recommends exercises which the student should do next. It is displayed whenever a registered student clicks on start in the welcome page abwillk.htm or selects "exercise wizard" from the menu.

...

...

Work File	Comments
abfrage.htm	It contains a questionaire for evaluating the tutorial system. The student who answers the questions does not have to enter an email address, so he can choose whether the mail is sent anonymously or not.
abframe.htm	The start page of the tutorial system. It defines four frames within the browser window: "lightgrey" is reserved for the table of contents, "darkgrey" for help texts and glossary terms, "black" for menu items and "white" for exercises.
abgloss.htm	If abframe.htm is loaded it will also display abgloss.htm which contains an introductory text for the "darkgrey" frame .
abhilfe.htm	It contains a case–insensitive help page which is displayed if the file in the white frame is not recognized by the JavaScript function, which is called when the help link is selected by the student.
abinhal.htm	It contains the table of contents of the tutorial system. Modifications should only be made by using the Tootsie development system.
abmenue.htm	It displayes the menu items of the tutorial system.
abnewus.htm	A new student must sign–in first before he can use the tutorial system. However, he is not only asked to enter his name, but also to answer a few questions in order to derive his current knowledge state. For each ticked box the user gets a certain amount of points as defined in the variable level. The final result is divided by the maximum amount of points and the difficulty level for the following exercises is set accordingly.
abwillk.htm	It contains the first page of the tutorial system. Basically it is responsible to reset the Cookie values for a new session.
hi*.htm	These files contain the case–sensitive help text of the tutorial system.
nw*.htm	These files are used by the newsgroup program. The file nwdummy.htm is needed to force the browser to display the news input form after the list of posted messages has been loaded.

.

B.2.2 Add–On

Add–ons are programs which are not essential for the operation of the tutorial system itself, however they can support the student in problem solving. For example, if a pocket calculator is integrated into the tutorial system, the student will not have to leave the work area, so consequently distractions can be avoided.

Table B.4: Add–On Files of the Tootsie Tutorial System.

Add–On File	Comments
stdvert.htm	A sample tool that allows the students to look up the values of the standard distribution.
taschre.htm	A sample pocket calculator with the arithmetic functions +, -, *, and \.

Appendix C

Example for Creating an Exercise

The following example shows the various steps which are required for generating an exercise with the Tootsie development system. For a detailed description of the input forms and their elements see Chapter 4.2, however it is recommended to start the development system and try the examples.

C.1 Generate Glossary

1. **Keyword:** `Diana`
   The keyword `Diana` is automatically referenced in an exercise text if it is delimited by white–space characters. As a consequence a hypertext link will not be set for a keyword which is followed by punctuation character like ".", ",", etc. The filename of a keyword consists of the following components: `g` for "glossary", the first four letters of the keyword, and a three digit number (for example, `gdian000.htm`).

2. **Title:** `Diana Tutorial System`

3. **Explanation**
   The last input box is used to describe the aforementioned keyword. The text must be structured by HTML commands.

```
<P ALIGN=LEFT>
   <A HREF="http://bayes.stat.washington.edu/diagnoser/diagnoser.html"
   TARGET="_top">Diagnostic Instructional Aid for Noetic Advancement</A>
</P>
```

After clicking on `Generate...` the glossary file is written, and the keyword is listed in the overview table.

C.2 Generate Exercise

The following example is based on the form "single–correct multiple–choice–question".

1. **Title:** `Bayes, 1. Question`
   From the section name `Bayes` a subdirectory `bayes` is created in which the exercise group 1. Question is stored. The filenames of an exercise group consist of the following components: `a` for "exercise", the first six letters of the group name (excluding white–space and punctuation characters), and the difficulty level (for example, `a1quest1.htm`).

2. **Glossary keywords:** `Diana`
   Glossary keywords are entered in form of a comma–separated list which specifies the terms that the development system must look for in the exercise text. If these terms are found, hypertext links will automatically be set to the files containing the glossary definitions.

3. **Help text**
   The format for entering help texts is `#<levels>#<text>`. If the input box is left empty by the system developer, a help button will not be displayed in the current exercise.

```
#1#
<P ALIGN=LEFT>This is the help text for help level 1.</P>
#23#
<P ALIGN=LEFT>This is the help text for help levels 2 and 3.</P>
#5#
<P ALIGN=LEFT>This is the help text for help level 5.</P>
```

   For help level 4 a help text is not defined, so instead a default text, which must be set in the resource file, is used. The help filenames consist of the following components: `h` for "help", the first six letters of the exercise group name, and the help level (for example, `h1quest1.htm`). This also means that for every page of an exercise group the *same* help texts must be entered as otherwise the existing help files will be overwritten (please note that help texts are defined for an exercise group, but not for a single exercise).

4. **Exercise text**
   An exercise text, which is structured with the help of HTML commands, must be entered. If the aforementioned glossary keyword `Diana` appears inside the text, a hypertext link will automatically be set to the appropriate glossary file.

5. **Answers**
   Up to five answers can be entered from which the student will have to select one. The correct answer must be ticked by the system developer.

6. **Difficulty level:** `1 (trivial)`

7. **Marked:** `yes`
   Marked exercises are stored in the Cookie variables and can be controlled by the tutorial system. Typical examples are:

   - `tootsie_info`, for example: `acbd`
     Each exercise group is represented by a single letter, whose character value encodes which difficulty levels of an exercise group are correctly solved by the student. If a learner accesses the tutorial system for the first time, the letters of the marked exercise groups are set to `a`. Whenever an exercise is solved from that moment on, its difficulty level specifies which bit of a binary word of five bits is set to `1`. The word and the binary value of the current letter are connected by the logical connective `or`, and the result is again stored in the Cookie variable `tootsie_info`. According to the example the difficulty levels 1 and 2 of the exercise group 2, i.e. `c`, are correctly solved by the student.

   - `tootsie_nexe`, for example: `aceabccdaa`
     A block of five letters represents one exercise group, where each difficulty level of the group, i.e. a single exercise, is symbolised by one letter. At the beginning of a course all letters are set to the default value `a`. Whenever the user accesses an exercise, the binary value of the letter is increased by one and again stored in `tootsie_nexe`. According to the example, from exercise group 1, i.e. `aceab`, the exercise with the difficulty level 2, i.e. `c`, was accessed three times, while the exercise with difficulty level 5, i.e. `b`, was only once loaded by the student.

After clicking on `Generate...` the exercise file is created and listed in the overview table of saved exercises.

C.3 Generate Links

Hypertext links must be set between exercises, so students can access the course pages. In this respect the overview table of saved exercises is used to copy exercise data into an internal clipboard by selecting a `Copy` link. This data is written into the entry fields whenever the system developers click on a command button.

1. **Exercise**
   After the system developer has selected the `Copy` link of the exercise `Bayes, 1. Question`[1], whose difficulty level is denoted by the superscript `1`, and clicked on `Page`, the text entry field contains the following data: `Bayes, 1. Question (single-correct, bayes/a1quest1.htm)`.

2. **Rules**
   The link rules specify which exercise is loaded next after the student has clicked on the `Continue...` button of the current exercise. Before a command button can be used the `Copy` link of an exercise must be selected, so the relevant data is stored in the internal clipboard of the development system. Only the buttons `Page` and `Group` set links to exercises, so each code section must end with a `link` command, as in the following example:

   ```
   if (correct)
      link(bayes/a11ques);
   else if (item[2])
      link(bayes/a11ques3.htm);
   else
   {
      reset(bayes/a11ques3.htm);
      link(bayes/a11ques3.htm);
   }
   ```

   If the student solves the current exercise `Bayes, 1. Question` a page from the exercise group `bayes/a11ques` is loaded, where the exact file depends on the current difficulty level. If the third answer is selected — the index numbers for `item` start with `0` — the page `bayes/a11ques3.htm` is presented to the learner. Otherwise the Cookie value of exercise `bayes/a11ques3.htm` is set from "correctly solved" to "not done yet" followed by a link to exercise `bayes/a11ques3.htm`.

3. **Comments**
   The format for entering comments and feedback, which are displayed on consecutive pages of the current exercise, is: `#link(<filename>);#<text>`. The `Page` button is again responsible for placing a `link` between the `#` delimiters as the following example shows:

   ```
   #link(bayes/a1quest5.htm);#
   Comment displayed on page "Bayes, 1. Question (level 5)" if the current
   exercise is correctly solved.
   #link(bayes/a11ques3.htm);#
   Comment displayed on page "Bayes, 1.1. Question (level 3)" if the
   current exercise is correctly solved.
   ```

   The comments are *not* source–specific, so they are displayed whenever the previous exercise was correctly solved. Please note that this feedback method cannot be applied for exercises in which a wrong answer was given by the students. In this case "hints–and–feedback" exercises should be used instead.

Clicking on `Generate...` changes the specified exercise files.

C.4 Generate Table of Contents

The input form shows a single text entry box in which the structure of the table of contents can be entered in form of HTML commands. Content entries are added by selecting the `Copy` link of a listed exercise group and clicking on the `Page` command button. Example:

```
<P ALIGN=LEFT><FONT SIZE=1>The Bayesian Laws</FONT></P>
link(bayes/a1quest);
link(bayes/a11ques);
<P ALIGN=LEFT><FONT SIZE=1>End</FONT></P>
```

In the resulting table of contents the `link` lines are replaced by a radio button and an abbreviated group name of the specified exercise group.

Appendix D

Glossary

ACT
Adaptive control of thought.

API
Application programming interface.

AWT
Abstract windowing toolkit.

behaviorism
"[...] a movement in psychology that advocates the use of strict experimental procedures to study observable behavior (or responses) in relation to the environment (or stimuli) ([Softkey Interational, 1996])".

CGI
Common gateway interface.

CL–HTTP
Common Lisp hypertext transfer protocol.

CLIM
Common Lisp interface manager.

CLOS
Common Lisp object system.

constructivism
Knowledge is not a representation of an external reality, but the result of perception. Constructivism emphasizes the active interpretation of objects by the student, and the meaningful construction of knowledge by learning–by–doing and criticism than by listening (see [Schulmeister, 1997], pp73/74).

CORBA
Common object request broker architecture.

CSS
Cascading style sheets.

DIANA
Diagnostic instructional aid for Noetic advancement.

ELM–ART
Episodic learner model adaptive remote tutor.

epistemology
"[...] the study or a theory of the nature and grounds of knowledge esp. with reference to its limits and validity ([Softkey Interational, 1996])".

GUI
Graphical user interface.

hermeneutical
"[...] interpretative ([Softkey Interational, 1996])".

HTML
Hypertext mark–up language.

HTTP
Hypertext transfer protocol.

JDBC
Java database connectivity.

MIME
Multipurpose internet mail extension.

M.I.T.
Massachusetts Institute of Technology.

OLE
Object linking and embedding.

POP
here: PUSH operational prototype.

PUSH
Plan and user sensitive help.

reification
The process or result of regarding something abstract as a material or concrete thing (see [Softkey Interational, 1996]).

SDP
System Development Process.

serendipity
"[...] the faculty or phenomenon of finding valuable or agreeable things not sought for ([Softkey Interational, 1996])".

SQL
Systematic query language.

URL
Universal resource location.

W3C
World–wide web consortium.

W3P
World–wide web presentation system.

WWW
World–wide web.

Appendix E

Figures

These two figures are actual screenshots of the Tootsie development system and a sample tutorial system. The latter is similar to the implementation mentioned in Section 4.3.3. Both screenshots were taken from Netscape Navigator 3.04 running on a Hewlett–Packard Series 700 workstation under HP–UX B.10.20, so the layout of the system depends on the used system configuration and world–wide web browser version. They may differ from other platforms.

In general, the layout and the user interface of the Tootsie system components are similarly designed, so all the available forms and exercise screens, which are described in Sections 4.2 and 4.3 in detail, are not presented.

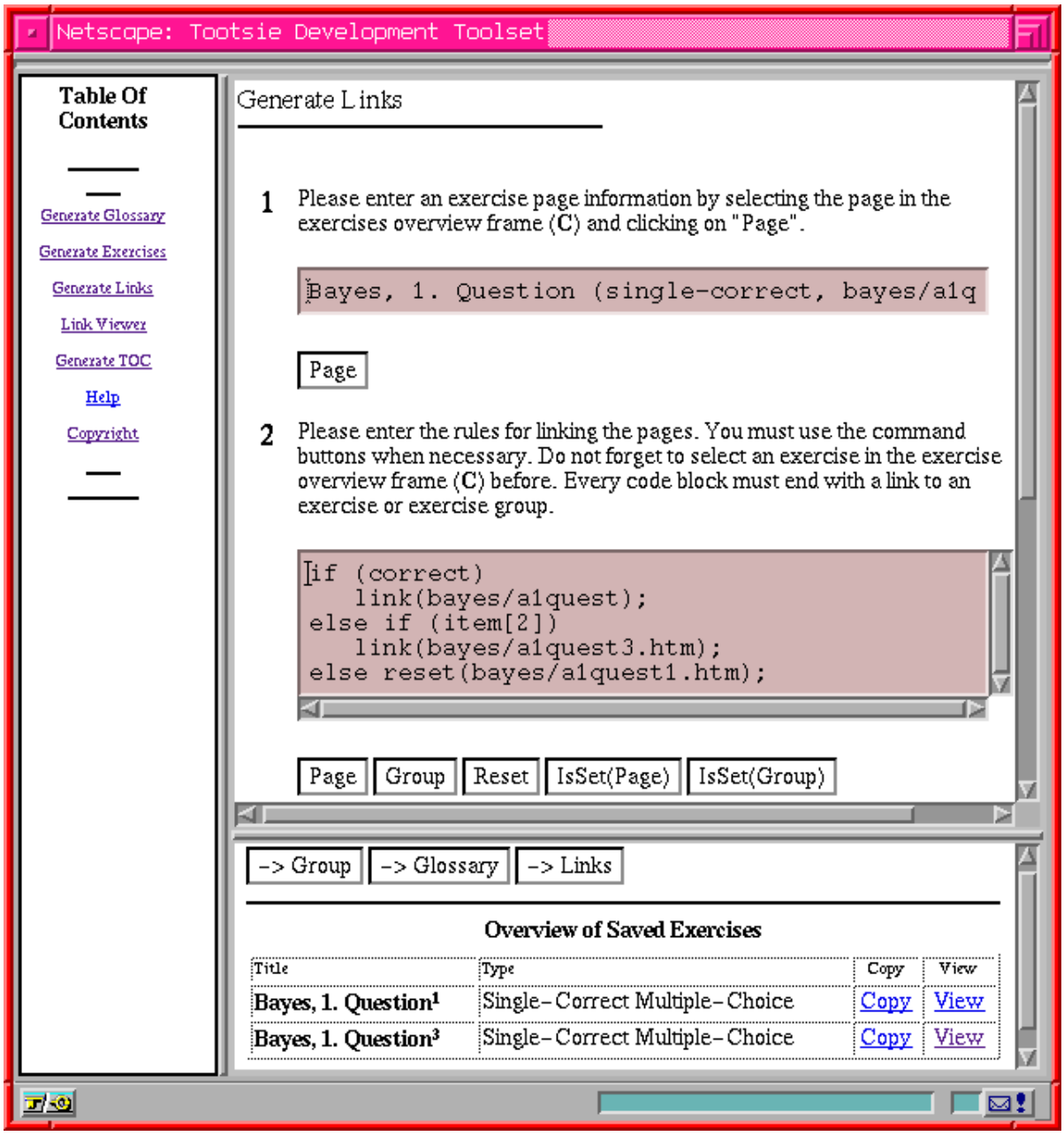


Figure E.1: The exercises of a course must be linked together to form a coherent information space. For that the `Copy` link of the overview table is especially important, because in conjunction with the predefined command buttons the correct data is then filled into the entry fields. The example shows that any JavaScript code can be entered into the "rules field": if the student correctly solves the exercise `Bayes, 1. Question`, a page from the exercise group `bayes/a1quest` is loaded depending on the current difficulty level. Otherwise, if the third answer is selected — the index numbers for `item` start with 0 — the page `bayes/a1quest3.htm` is next, or the Cookie for exercise `bayes/a1quest1.htm` is set from "correctly solved" to "not done yet".

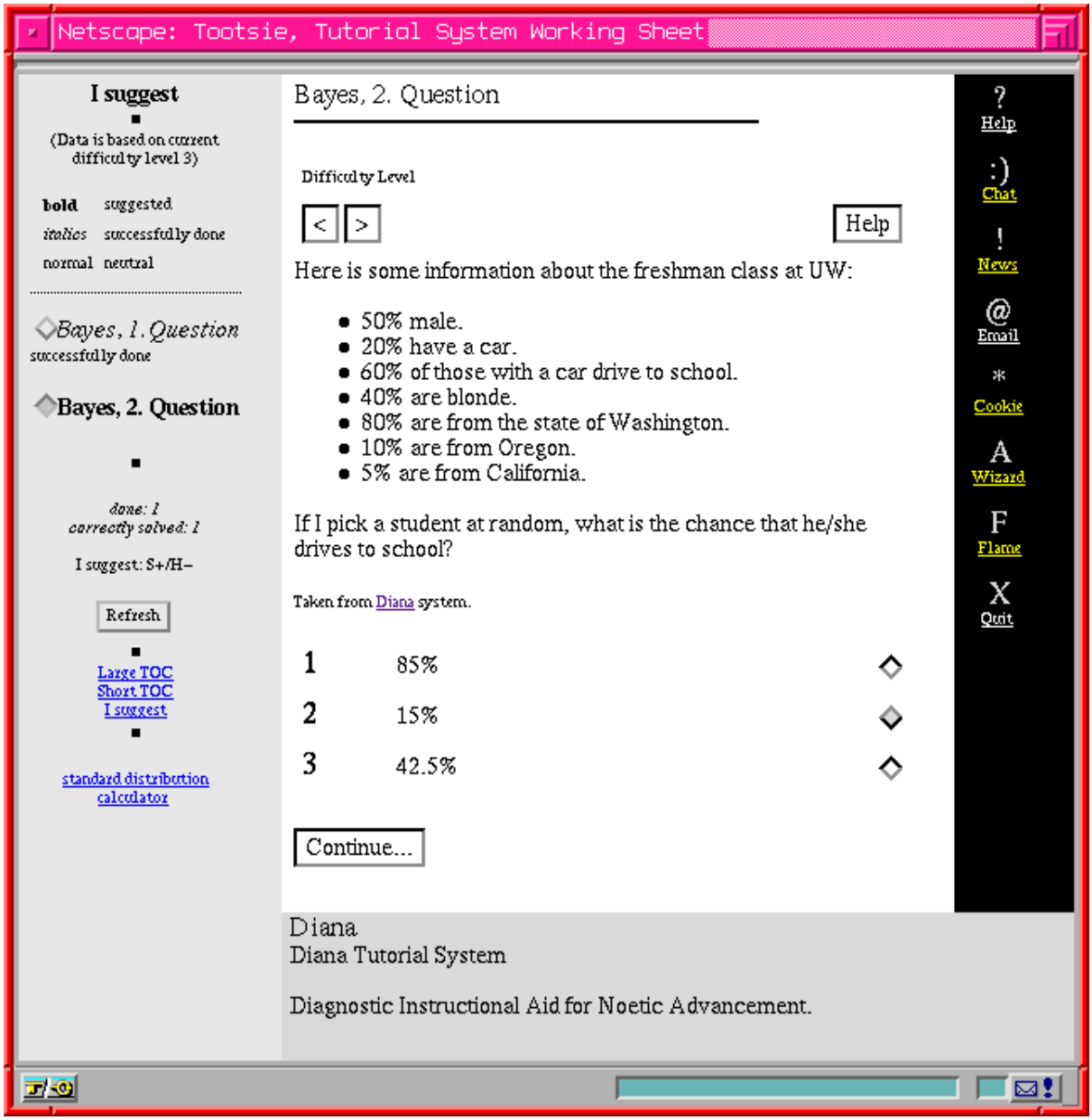


Figure E.2: This single–correct single–choice exercise offers three possible answers to the student. By clicking on `Continue...` the selected answer is processed, and the next page is loaded according to the JavaScript source code which links the different course documents. The underlined "Diana" hypertext link marks a glossary keyword, whose definition is displayed in the lower section of the browser window. In addition, a help page for the current exercise is available because a `Help` button is shown in the upper right corner of the white frame. The two buttons under "Difficulty Level" respectively decrease or increase the difficulty level. This will however only effect the following exercises, provided that an exercise for the selected difficulty level exists. The table of contents is set to `annotated` or `I suggest`, so the tutorial system recommends exercises to the learner with the help of different font styles. These styles can also be combined: for example, an exercise title written in bold and italic face denotes a page which was once solved by the student, but wrongly answered in the following sessions. The system therefore assumes that this exercise is especially difficult for the learner.

Bibliography

[Anderson, 1996]
Anderson J. R., *Kognitive Psychologie, 2. Auflage.*
Spektrum Akademischer Verlag, Heidelberg, 1996.

[Anderson et al., 1995]
Anderson J. R., Corbett A., Koedinger K., Pelletier R., *Cognitive Tutors: Lessons Learned.*
http://act.psy.cmu.edu/ACT/papers/Lessons_Learned–abs.html, 1995.

[Andrews, 1996]
Andrews K., *HyperWave: The Next Generation Web Server.*
http://wksun2.wk.or.at:8000/0x811b9908_0x00251dce;sk=620A0EA1, 1996.

[Astleitner, 1996]
Astleitner H., *Lernen in Informationsnetzen.*
Habilitationsschrift, Institut für Erziehungswissenschaften, Salzburg, 1996.

[Asymetrix, 1997]
Asymetrix Toolbook, *Features of Asymetrix Toolbook.*
http://www.asymetrix.com/products/toolbook2/, 1997.

[Brusilovsky & Pesin, 1996]
Brusilovsky P., Pesin L., *ISIS–Tutor: An Intelligent Learning Environment for CDS/ISIS Users.*
http://cs.joensuu.fi/~mtuki/www_clc.270296/Brusilov.html, 1996.

[Burns, 1997]
Burns J., *So You Want to Layer, Huh?*
http://www.htmlgoodies.com/, 1997.

[December Communciatons, 1997]
December Communications Inc., *Environment Variables for Use in Gateway Programming.*
http://www.december.com/html/spec/envvars.html, 1997.

[December, 1997]
December J., *Level 4 HTML Summary.*
http://www.december.com/html/spec/level4.html, 1997.

[Digital Think, 1997]
Digital Think Orientation, *The DigitalThink Training Method.*
http://www.digitalthink.com/, 1997.

[Döring, 1996]
Döring N., *Lernen und Lehren im Netz.*
http://www.cs.tu–berlin.de/~doering/, 1996.

[Eastmond & Granger, 1997]
Eastmond D., Granger D., *Reaching Distance Students with Computer Network Technology (Part I).*
http://distance–educator.com/Reaching–1.2.html, 1997.

[Eberl & Jacobsen, 1997]
Eberl M., Jacobsen J., *Macromedia Director 5 für Insider.*
SAMS, Haar bei München, 1997.

[Espinoza & Höök, 1996]
Espinoza F., Höök K., *An Interactive WWW Interface to an Adaptive Information System.* http://www.sics.se/~espinoza/, 1996.

[Espinoza, 1996]
Espinoza F., *A World Wide Web Based Presentation System For An Adaptive Help System.* Uppsala University Computer Science Department, Uppsala, 1996.

[Faber, 1993]
Faber W., *Hypermediale Lernsysteme.* http://aia.wu–wien.ac.at/Publikationen/Faber/WU–JT.html, 1993.

[Flanagan, 1996]
Flanagan D., *Java in a Nutshell.* O'Reilly & Associates Inc., Camebridge, 1996.

[Furman & Isaacs, 1997]
Furman S., Isaacs S., *Positioning HTML Elements with Cascading Style Sheets.* http://www.w3.org/TR/WD–positioning–19970819, 1997.

[Gonschorek, 1997]
Gonschorek M., *Intelligente Lehrsysteme — Ein Überblick.* Institut für Informatik, Informatik I (Hauptseminar Intelligente Lehrsysteme), Technische Universität München, München, 1997.

[Hall et al., 1996]
Hall W., Davis H., Hutchings G., *Rethinking Hypermedia — The Microcosm Approach.* Kluwer Academic Publishers, Dordrecht, 1996.

[Handke, 1997]
Handke J., *Multimedia mit ToolBook und Macromedia Director.* Oldenbourg, München, 1997.

[Harrer, 1996]
Harrer A., *Ein didaktisches Konzept für die Lernerführung in einem intelligenten Lehrsystem.* Institut für Informatik, Informatik I (Diplomarbeit), Technische Universität München, München, 1996.

[Heath, 1996]
Heath S., *Multimedia & Communications Technology.* Focal Press, London, 1996.

[Herzog, 1996]
Herzog C., *SYPROS: Ein intelligentes Lehrsystem für die Synchronisation paralleler Prozesse mit Semaphoren.* Institut für Informatik, Informatik I (Kolloquiumsvortrag Duisburg), Technische Universität München, München, 1996.

[Höök, 1996]
Höök K., *Plan– and User Sensitive Help (P.U.S.H.).* http://www.sics.se/uacm/push.html, 1996.

[Hüskes, 1997]
Hüskes R., *Schnittmuster für Web–Schneider.* c't 1997, Heft 12 (pp240–245), Hannover, 1997.

[Internet Engineering Task Force, 1997]
Internet Engineering Task Force, *Hypertext Transfer Protocol HTTP/1.1 (draft).* http://www.w3.org/Protocols/History.html, 1997.

[Kaiser & Kaiser, 1994]
Kaiser A., Kaiser R., *Studienbuch Pädagogik – Grund– und Prüfungswissen, 7. Auflage.* Cornelsen Scriptor, Frankfurt, 1994.

[Keith, 1997]
Keith D., *LISP Lecture Notes.*
http://www.ifi.ntnu.no/~keithd/classes/lisp/lectures/l1/index.htm, 1997.

[Kleinschroth, 1996]
Kleinschroth R., *Neues Lernen mit dem Computer.*
rororo, 1996.

[Kopka, 1994]
Kopka H., *LaTeX Einführung Band 1.*
Addison–Wesley (Deutschland), Bonn, 1994.

[Lai et al., 1995]
Lai M., Chen B., Yuan S., *Toward A New Educational Environment.*
http://www.w3.org/Conferences/WWW4/Papers/238/, 1995.

[Lie & Bos, 1996]
Lie H., Bos B., *Cascading Style Sheets –Level 1.*
http://www.w3.org/pub/WWW/TR/REC–CSS1, 1996.

[Madigan et al., 1995]
Madigan D., Clarkson D., Donnell D., Hunt E., Keim M., Minstrell J., Nason M., Schaffner A., Volinsky C., *Facet–based Learning for Statistics.*
http://www.stat.washington.edu/andrew/fbl.html, 1995.

[Mallery, 1997]
Mallery J., *Common Lisp HTTP Server Homepage.*
http://www.ai.mit.edu/projects/iiip/doc/cl–http/home–page.html, 1997.

[Mallery, 1994]
Mallery J., *A Common LISP Hypermedia Server.*
Proceeding of The First International Conference on The World–Wide Web, Geneva, 1994.

[Netscape Developer, 1997]
Netscape Developer, *General Developer Documentation.*
http://developer.netscape.com/library/documentation/index.html, 1997.

[Netscape Developer, 1997a]
Netscape Developer, *JavaScript Documentation.*
http://developer.netscape.com/, 1997.

[Netscape Developer, 1997b]
Netscape Developer, *Plug–in Basics.*
http://developer.netscape.com/library/documentation/communicator/plugin/, 1997.

[Netscape Developer, 1997c]
Netscape Developer, *Dynamic Documents.*
http://developer.netscape.com/library/documentation/communicator/dynhtml/index.htm, 1997.

[Netscape Developer, 1997d]
Netscape Developer, *LiveConnect.*
http://home.netscape.com/comprod/products/navigator/version_3.0/building_blocks/liveconnect/how.html, 1997.

[Netscape Developer, 1997e]
Netscape Developer, *Persistent Client State HTTP Cookies.*
http://developer.netscape.com/library/documentation/index.html, 1997.

[Nistor & Mandl, 1995]
Nistor N., Mandl H., *Lernen in Computernetzwerken. Erfahrungen mit einem virtuellen Seminar (Forschungsbericht 64).*
Ludwig–Maximilians–Universität, Lehrstuhl für Empirische Pädagogik und Pädagogische Psychologie, München, 1995.

[Ping–Jer et al., 1996]
Ping–Jer Y., Bih–Horng C., Ming–Chih L., Shyan–Ming Y., *Synchronous Navigation Control for Distance Learning on the Web.*
http://www5conf.inria.fr/fich_html/papers/P28/Overview.html, 1996.

[Polson & Richardson, 1988]
Polson M. C., Richardson J. J., *Foundations of Intelligent Tutoring Systems.*
Lawrence Erlbaum Associates Publishers, Hillsdale NJ, 1988.

[Powersim Corporation, 1997]
Powersim Corporation, *Powersim Metro JX.*
http://www.powersim.no/, 1997.

[Raggett et al., 1997]
Raggett D., Le Hors A., Jacobs I., *HTML 4.0 Specification.*
http://www.w3.org/TR/WD–html40–970917/, 1997.

[Reinhardt & Schewe, 1995]
Reinhardt B., Schewe S., *A Shell for Intelligent Tutoring Systems.*
http://ki–server.informatik.uni–wuerzburg.de/HTMLs/ls6–info/Publikationen/95/Reinhardt–AI–ED95/Reinhardt–AI–ED95.doc.html, 1995.

[Reinmann–Rothmeier & Mandl, 1995]
Reinmann–Rothmeier G., Mandl H., *Auf dem Weg ins Informationszeitalter? Was Wirtschaft, Politik und Öffentlichkeit bewegt und auf die Bildung zukommt (Forschungsbericht 54).*
Ludwig–Maximilians–Universität, Lehrstuhl für Empirische Pädagogik und Pädagogische Psychologie, München, 1995.

[Sackl, 1997]
Sackl R., *Computergestützte Kommunikation für verteilte Lerngruppen in der Vorlesung 2000 Umgebung und prototypische Implementierung in Java.*
Institut für Informatik, Informatik XI (Diplomarbeit), Technische Universität München, München, 1997.

[Schaffner et al., 1996]
Schaffner A., Madigan D., Donnell D., Hunt E., Keim M., Minstrell J., Nason M., Volinsky C., *Benchmarks, Facets and the World–Wide Web: Tools for the Advancements of Undergraduate Statistics Education.*
http://www.stat.washington.edu/andrew/fbl.html, 1996.

[Schulmeister, 1997]
Schulmeister R., *Grundlagen hypermedialer Lernsysteme, 2. Auflage.*
Oldenbourg, München, 1997.

[Schult, 1996]
Schult T., *Computer Based Training.*
c't 1996, Heft 9 (p. 178–186), Hannover, 1996.

[Seidel, 1993]
Seidel C., *Computer Based Training.*
Verlag für angewandte Psychologie, Verlagsgruppe Hogrefe, Göttingen, 1993.

[Softkey Interational, 1996]
Softkey International, *Infopedia 2 — The Ultimate Multimedia Encyclopedia and Reference Library.*
Softkey International, Infopedia Version Release R11, 1996.

[Spada, 1992]
Spada H. (publ.), *Allgemeine Psychologie, 2. Auflage.*
Verlag Hans Huber, Bern, 1992.

[Sun Microsystems, 1997]
Sun Microsystems Inc., *JDK 1.1.1 Documentation.*
http://java.sun.com/, 1997.

[Sun Microsystems, 1998a]
Sun Microsystems Inc., *What is Swing?*
http://java.sun.com/products/jfc/swingdoc–current/what_is_swing.html, 1998.

[Sun Microsystems, 1998b]
Sun Microsystems Inc., *Frequently Asked Questions — Java Security.*
http://java.sun.com/sfaq/index.html, 1998.

[Taubenberger, 1997]
Taubenberger M., *Entwicklung einer adaptiven Erklrärungskomponente für das intelligente Lehrsystem POINTRA unter Berücksichtigung von Lösungsvarianten.*
Institut für Informatik, Informatik I (Diplomarbeit), Technische Universität München, München, 1997.

[Weber & Specht, 1997]
Weber G., Specht M., *User Modeling and Adaptive Navigation Support in WWW–based Tutoring Systems.*
Proceeding of User Modeling '97, Cagliari, 1997.

[Weinert & Mandl, 1997]
Weinert F., Mandl H. (publ.), *Psychologie der Erwachsenenbildung. Enzyklopädie der Psychologie, Band 4, 11. Kapitel.*
Verlagsgruppe Hogrefe, Göttingen, 1997.

[Welsch, 1996]
Welsch N., *Entwicklung von Multimedia–Projekten mit Macromedia Director und Lingo.*
Springer–Verlag, Berlin, 1996.

[Wenger, 1987]
Wenger E., *Artificial Intelligence and Tutoring Systems.*
Morgan–Kaufmann, Los Altos, 1987.

Reinhard Schaffner

Concepts for the Implementation of Tutorial Systems in HTML and Java

Diplomarbeit
1998

Technische Universität München
Fakultät für Informatik