CHAPTER 25: User Modeling

Addie Johnson[1] and Niels Taatgen[2]

University of Groningen

[1]Department of Experimental and Work Psychology; [2]Department of Artificial Intelligence

CHAPTER 25: User Modeling

Predictive Models....3
Models of Task Performance....6
Processing Architectures that Interact with the Outside World....7
Architectures that Incorporate Learning....9
Hybrid Architectures....11
Personalized Models....16
Determining User Characteristics....17
Updating the User Model on the Basis of Interactions....19
Adaptive Elements....21
Applications....24
Intelligent Tutors....24
Facilitating Interaction Among Users....28
Electronic Books....28
Agents....30
Agents as Team Members....33
User Modeling Servers....35
Adapting to Physical Limitations....37
Summary....37
References....39
Table 25-1. Overview of Constraints that the Model Human Processor (MHP), Soar, EPIC and ACT-R Impose on Cognitive Processing....45
Table 25-2. Adaptable Aspects of Web Use....48
Table 25-3. Functions of a User-modeling System....49
Figure Captions....50

CHAPTER 25: User Modeling

The worldwide web had an estimated 655 million users in 2002 (http://cyberatlas.internet.com/big_picture/geographics) and was accessed by people of essentially all possible backgrounds. Each of these users had a goal in mind, whether it be trying to book a flight, search for information on a research topic, or just while away a few hours. Different users also have different knowledge, interests, abilities, learning styles, and preferences regarding information presentation. An increasingly important research area is how interfaces can be designed to recognize the goals and characteristics of the user, and adapt accordingly.

Companies, universities, and other organizations are becoming increasingly aware of the need to personalize web pages for individual users or user groups. In order to offer personalized information, it is necessary to monitor a user's behavior and to make generalizations and predictions based on these observations. Information about the user that can be drawn on in this way is called a *user model* (see Fischer, 2001, for a review).

Modeling the user may be as simple as fitting a *user profile* (e.g., single, young, female) or as complicated as discovering expert knowledge (e.g., how a chemist would classify a data set). The modeling system may acquire information explicitly by means of a user-completed questionnaire or implicitly, by observing user actions and making inferences based on stored knowledge. The goal of user modeling may be to predict user behavior, to gain knowledge of a particular user in order to tailor interactions to that user, or to create a database of users that can be accessed by others. The goal of user modeling may even be the creation of the model itself, when that model is used to create an autonomous agent to fill a role within a system.

We begin this chapter by describing the major modeling systems commonly used to create predictive models of the user, that is, models that can be used to predict human

behavior in a human-machine system. All of the models we discuss in this section are characterized by constraints that limit the computational power of the model. The purpose of the constraints is to enable the modeler to build models that operate like humans—that is, within the same constraints. The models produced within these systems can be used to test theories about how people learn and perform cognitive tasks. They can also be used for practical purposes such as testing the usability of different human-computer interfaces or inferring the knowledge structure of the user so that appropriate remedial help can be supplied.

The models discussed in the first section of the chapter have as their goal the description of how people, in general, perform a task. In the second section of the chapter, we focus on the individual user. Our goal in this section is to describe techniques for gathering information about individual users and to describe how the computer interface can be adapted on the basis of that information. We end the chapter with a discussion of various applications, ranging from tutoring programs to autonomous intelligent agents.

Predictive Models

The usual method of testing a new interface design is to perform a user-evaluation study. Potential users are asked to carry out a number of tasks with the new interface and their performance is evaluated. Such user studies are typically time-consuming and costly. An alternative approach is to develop cognitive models of user performance and to use those models to predict behavior. Such models, or "synthetic users," have several advantages over human participants. First, once the model has been constructed, it can be used repeatedly to evaluate incremental changes in design. Second, a cognitive model may offer insight into the nature of the task. In a cognitive model, unlike in a human participant, each reasoning step can be traced, the contents of the model's memory can be inspected to see what the model has learned, and the errors made can be traced back to their source.

The generic user model that can be used to test any user interface is a holy grail of human factors. The current state-of-the-art is that models are developed for specific tasks, or aspects of tasks (e.g., menu search, icon identification, deployment of attention, or automatization), and then validated on a case-by-case basis using human data. Although no single generic model exists to test the complete range of applications, several attempts have been made to create models that can predict the outcomes of experiments rather than merely explaining outcomes after the experiments have been conducted (e.g., Salvucci & Macuga, 2001).

The first step in developing a predictive user model is to perform a task analysis (see Chapter 24 for a more detailed overview of methods of task analysis). A task analysis gives a specification of the knowledge that is needed to perform the task and the sequence of operations required. Although a task analysis gives an indication of the complexity of the task, it does not generally take into account the details of human information processing. More accurate user models can be created by augmenting the task analysis with a specification of the constraints on human information processing that should be satisfied within the model (see Table 25-1 for a summary of how the major predictive models do this). For example, the Model Human Processor (MHP; Card, Moran, & Newell, 1983), described below, provides a means of specifying the time to perform specific operations, the probability of errors, and speed-up due to learning for the sequence of operations specified in the task analysis.

A more advanced method of incorporating human information processing constraints in task models is to embed the models in an architecture of cognition, a simulation environment that can, given the necessary knowledge to do the task, mimic human behavior on that task. The knowledge specified in a task analysis within an architecture of cognition can be used to make predictions about various aspects of human performance, including

reaction times, errors, choices made, and eye movements. Note that an architecture of cognition is a simulation environment, but also a theory, and will sometimes be referred to as such. For example, architectures typically incorporate a theory about memory that specifies how knowledge is represented and how memory processes such as storage, retrieval, and forgetting function. This theory is embodied in the simulation environment and governs how the memory system will behave. The term "model" is used for simulations of specific tasks. It should be noted that only models can make specific predictions. Therefore, although the architecture may incorporate a theory of memory, a specific model of, for example, a digit span task, is needed to make predictions regarding the performance of such a task.

A problem of cognitive models is that it is not easy to assess their validity. In general, it is assumed that a model that produces the same behavior as people do is a valid model. However, several different models might produce the same behavior, in which case a different criterion is needed to determine which model is best. Unfortunately, there is no quantitative measure for model validity, but most cognitive modelers agree that the following qualitative factors determine the validity of a model:

- *A good model should have as few free parameters as possible*. Many cognitive architectures have free parameters that can be given arbitrary values by the modeler. Because free parameters enable the modeler to manipulate the outcome of the model, increasing the number of free parameters diminishes the model's predictive power
- *A model should not only describe behavior, but should also predict it* . Cognitive models are often made after the experimental data have been gathered and analyzed. A model with high validity should be able to predict performance.

A model should learn its own task-specific knowledge. Building knowledge into a model increases its specificity, and may decrease its validity.

Most of the current approaches to predictive modeling use task analysis to specify the knowledge that an expert would need to do the task. This violates the validity criterion stated above, that a model should acquire task specific knowledge on its own. Moreover, basing a model on a task analysis of expert performance means that the model is of an expert user whereas the typical user may not have mastered the task being modeled. Useful predictions and a complete understanding of the task requires that models be built that start at the level of a novice and gradually proceed to become experts in the same way people do. In other words, many applications require building models that not only perform as humans do, but that learn. Accordingly, after focusing on models in which expert task performance is central, we then move on to models of learning and models that learn on the basis of instructions. Most, although not all (e.g., LICAI; Kitajima & Polson, 1997), performance-based models are based on production rules (condition-action pairs). Because production-rule models are by far the most commonly used models in the field, we restrict our discussion to this sort of model.

Models of Task Performance

The starting point of a task-performance model is to investigate the knowledge needed to perform the task at an expert level. The GOMS (Card, Moran & Newell, 1983) technique, in which the task is analyzed in terms of goals, operators, methods, and selection rules, can be used for this end. *Goals* form a hierarchy starting from the top goal that represents achieving the end result and proceeding to the so-called unit tasks, which are sub-goals that cannot be further decomposed. For example, the top goal could be to edit a text, while a goal at the unit-task level might be to move the cursor to a certain line in the text. A general assumption is that unit tasks can be completed in the order of 10 seconds. In order to achieve a goal at the unit-task level, *methods* are needed to specify what actions have to be carried out in terms of the *operators* that perform the actions. A method for moving to a certain line in a text might specify applying the operator “press arrow key” until the desired line is reached. A different

method might specify the operator "key in <control-x> followed by the line number." When a choice must be made between alternative methods, *selection rules* are needed that specify when a certain method should be used (e.g., if the cursor is more than five lines away, use Method 2).

A GOMS analysis enables us to perform a cognitive walk-through of the task of interest. Given the main goal, we can specify the order in which sub-goals are posed and attained, which operators are used, and which choices are made to eventually achieve the main goal. At this level of analysis it is possible to describe the order in which the expert user will execute actions. This will, in general, not be of much interest given that the GOMS analysis itself is based on the behavior of the expert. In order to make more interesting predictions it is necessary to augment the GOMS analysis with a psychological model. The first psychological model to be used in conjunction with GOMS was the Model Human Processor (MHP; Card, Moran, & Newell, 1983). Figure 25-1 shows a simplified version of the MHP, comprising a memory system (working memory and long-term memory) and three processors (perceptual, cognitive, and motor). Each of the processors is assigned an approximate cycle time. For example, the motor processor needs on the order of 100 ms to prepare a motor response. Working memory has a limited capacity (about 7 elements) and a limited retention time (about 7 seconds). The combination of the MHP and GOMS can be used to make simple predictions about quantitative aspects of human performance. The MHP can be used to annotate the analysis that is made with GOMS by supplying approximate execution times. It can also trace working-memory usage and signal potential capacity problems.

Processing Architectures that Interact with the Outside World

The MHP approach is useful, but also severely limited. It can only approximately predict task execution times, and is vague regarding what can be done in one processing

cycle. More precise and constrained predictions can be made with the more elaborate theoretical framework of the EPIC (Executive Process-Interactive Control) architecture (Kieras & Meyer, 1997). Contrary to GOMS/MHP, EPIC allows the implementation of processing models, simulations of the user that can be run on a computer and that can be used to prove the soundness of the analysis and to provide a concrete prediction of task performance. The main theoretical goal in a theory such as EPIC is to constrain all possible simulations of behavior to only the behavior exhibited by users. As can been seen in Table 25-1, EPIC's main source of constraints is in the perceptual-motor aspects of the task, whereas central cognition is relatively unconstrained. The perceptual-motor modules in EPIC can handle only a single action at a time, and each of these actions take a certain amount of time. Although a module can do only one thing at a time, expert behavior on a task is exemplified by skillful interleaving of perceptual, cognitive, and motor actions. EPIC's modules incorporate mathematical models of the time it takes to complete operations that are based on empirical data. The knowledge of the model is represented using production rules. A production rule consists of a set of conditions that is tested against the current internal state and state of the modules, and a set of actions that is carried out once all conditions are satisfied. Production rules are a fairly universal way of representing knowledge: Although the exact details of the syntax differ, almost all cognitive architectures use production rules.

EPIC has been applied in a number of contexts. For example, Hornof and Kieras (1997; described in Kieras & Meyer, 1997) applied EPIC to menu search. The task modeled was to find a label in a pull-down menu as quickly as possible. Perhaps the simplest model of such a task is the serial-search model in which the user first attends to the top item on the list and compares it to the label being searched for. If the item does not match the target, the next item on the list is checked; otherwise, the search is terminated. EPIC's predictions of search time using this method can be obtained by describing the strategy in EPIC production rules

and performing a simulation in a test environment in which menus have to be searched. As shown in Figure 25-2, such a model overestimates actual search time (obtained with human subjects), except when the target is in the first position to be searched.

In an alternative model, the overlapping search model, the parallelism of the cognitive system is exploited. Instead of waiting for the cognitive system to finish deciding whether or not the requested label is found, the eye moves on to the next item in the list while the first item is still being evaluated. Such a strategy results in the situation that the eye has to move back to a previous item in the list once it has been decided that the item has been found, but this is a small price to pay for the speed-up this parallelism produces (see Figure 25-2). Parallelism is allowed in EPIC as long as no two perceptual-motor modules are used at the same time. In practice, the most influential constraint is posed by the duration of actions. For example, in the serial-search model, the parameter that influences the slope could, in theory, be changed to make this (incorrect) model match the data. EPIC precludes this from occurring because an eye-movement takes a certain amount of time, as does a decision as to whether the label is correct or not, such that the data can only be explained if these actions occur in parallel.

Architectures that Incorporate Learning

As a cognitive architecture, EPIC states that the main sources of constraints on human performance are the perceptual and motor aspects of the task. However, the theory is too flexible to be able to predict task performance before any data has been gathered. For example, although the theory allows a certain degree of parallelism, it cannot predict a priori whether this possibility will be exploited in a given task. One approach to developing a stronger theory capable of specifying the most plausible model of task performance is to incorporate learning mechanisms that make it possible for task models to emerge from the cognitive architecture, rather than relying on knowledge supplied by the modeler.

An example of a learning architecture is Soar (States, Operators, and Results; Newell, 1990). In Soar, new knowledge is learned when impasses are encountered during problem solving. These impasses often take the form of a choice in which there are several possible actions available and no clear decision rule for selecting the appropriate one. Faced with such a choice problem, Soar will evaluate each of the possible actions, and will select the best one. The by-product of this evaluation process is a new rule that will, when Soar is faced with a similar situation, enable the correct choice to be made without invoking evaluation processes.

An example of a Soar model in the domain of menu search is Ayn (Howes, 1994). Ayn models the task of finding an item in a typical drop-down menu bar containing several labeled menus each of which contains several items. The model makes predictions regarding how many items will have to be searched before the correct item is found. Initially, the model searches all menus exhaustively. However, during search it generates new rules that are subsequently used to decrease search times. Eventually, the model learns exactly where to look for each menu item. In one example, the model has to search for a menu option that formats a document as two columns. The model starts out with no knowledge of where to find the information, and thus is faced with an impasse: There are a number of potentially applicable menus, but the model cannot choose between them. As a consequence, the model simply tries out the options, successively, exploring all options until the correct one is found. For example, the “file” menu might be unsuccessfully explored before the correct choice (“section”) is found under the “format” menu. After successfully achieving the goal, Soar learns two new production rules: A rule that specifies the incorrectly chosen menu as the wrong one for the goal of formatting the document as two columns, and one that specifies the correct sub-menu. These two rules are insufficient for a direct solution of the two-column format problem because there is no rule that says that the format menu is the right menu, only

that the file menu is the wrong one. Only after more practice will the model learn to find the two-column option without first exploring irrelevant options.

Soar's learning is purely symbolic and not subject to forgetting. A Soar model would therefore predict (probably incorrectly) that finding the two-column option would occur flawlessly after it has been mastered, even if a year passes by in the meantime. Another limitation of the Soar approach is that its learning is tied to impasses. This seems at odds with the phenomenon of implicit learning (Reber, 1989), in which new knowledge may be acquired without the problem solver being conscious of it. Forgetting is accounted for in modeling approaches that attach numeric quantities (e.g., "activation") to the knowledge elements in the architecture, an approach that is sometimes referred to as *sub-symbolic representation*. Architectures that use both symbolic and sub-symbolic representations are referred to as hybrid architectures.

Hybrid Architectures

ACT-R (Adaptive Control of Thought-Rational; Anderson, 1993; Anderson & Lebiere, 1998) is an example of a hybrid architecture that supports learning and sub-symbolic computations. The core of the ACT-R architecture is a production system similar to that of EPIC and Soar. This production system core is surrounded by a set of modules that are similar to the perceptual and motor modules used by EPIC[1]. ACT-R also has a declarative memory that is used to store facts. Facts in declarative memory have activation values that reflect how often a fact has been used in the past, and its association with the current context. Activation determines how much time it takes to retrieve a fact from memory and whether it can be retrieved at all. All the modules communicate with the production system through buffers: a visual buffer, a manual buffer, and a retrieval buffer (declarative memory). In addition to the

[1] The current version of ACT-R incorporates ACT-R/PM (Byrne & Anderson, 2001) an extension to ACT-R that implements the same perceptual/motor modules EPIC has.

buffers that correspond to modules, there is a goal buffer that has no corresponding modules, but that is used to hold the current goal. Production rules have a utility value associated with them that reflects the success of the rule in the past in terms of time cost, and success and failure. On each cycle of the production system, the rule with the highest utility is chosen from the rules that match the current contents of the buffers.

Byrne (2001) developed an ACT-R model of menu search on the basis of eye-movement data collected during menu search tasks. These data suggest that people do not use a single strategy to search menus, as predicted by previous models. Byrne modeled this by incorporating several competing strategies in the model, and by using ACT-R's utility learning mechanism to determine which strategy should be used. Byrne's model is a good example of a case where both learning and subsymbolic computation are needed to explain the full breath of behavior. Another example of the importance of sub-symbolic computation is SNIF-ACT by Pirolli and Fu (2003). SNIF-ACT is a model of searching for information on the worldwide web. Given a certain starting page, the model attempts to select the best link to reach its goal. It does this by picking the link with the strongest "information-scent," that is, the link that receives the highest activation through spreading activation from the goal information. If the information-scent is not strong enough, the model will abandon the current page, and backtrack to earlier pages. The model can be used to evaluate the quality of websites: If SNIF-ACT is not able to find the desired information, users probably will not find it, either.

A more elaborate illustration of the type of modeling possible with ACT-R is based on a simplified air traffic control task (KA-ATC; Ackerman, 1988). The model of the task is explained in detail in Taatgen (2002) and Taatgen and Lee (2003). In this task, participants land planes by choosing a plane that is waiting to be landed and designating the runway on which the plane should land. There are four runways, the use of which is restricted by rules

that relate to the length of the runway, the current weather, and the type of plane that is to be landed. For example, a DC10 can only be landed on a short runway if the runway is not icy and the wind is below 40 knots. Although participants receive an extended instruction period, they tend to forget some rules—especially the more complicated ones regarding weather, plane type, and runway length. The goal of Taatgen's ACT-R model is to capture the learning in this task by predicting the improvement in performance of the participants at both a global level and at the level of individual keystrokes.

An example of a production rule from the air traffic control task is:

IF The goal is to land a plane and a plane has been selected that can be landed on the short runway (match of goal buffer)

AND you are currently looking at the short runway and it is not occupied (match of visual buffer)

AND the right hand is not used at this moment (match of manual buffer)

THEN note that we are moving to the short runway (change to goal buffer)

AND push the arrow-down key (change to manual-buffer)

AND move attention to the weather information (change to visual buffer)

This rule reflects the knowledge of an expert on the task at the stage in which a plane has been selected that has to be directed to the short runway. After checking whether the short runway is available, it issues the first motor command, and also initiates an attentional shift to check the weather, information regarding which might be needed for landing the next plane.

Although this example rule is very efficient, it is also highly task-specific; rules like this have to be learned in the process of acquiring the skill. For novices, the model assumes that all the task-specific knowledge needed about air traffic control is present in declarative memory, having been put there by the instructions given to participants. This knowledge has a low activation because it is new, and might have gaps in it in places where the participant did

not properly memorize or understand the instructions. The production rules interpret these instructions and carry them out. Two examples of interpretive rules are:

Get-next-instruction-rule

IF the goal is to do a certain task and you have just done a certain step (goal buffer)

THEN request the instruction for the next step for this task (retrieval buffer)

Carry-out-a-push-key-rule

IF the goal is to do a certain task (goal buffer)

AND the instruction is to push a certain key (retrieval buffer)

AND the right hand is available (manual buffer)

THEN note that the instruction is carried out (goal buffer)

AND push the key (manual buffer)

A characteristic of interpreting instructions is that it results in behavior that is much slower than that of experts: Retrieving the instructions takes time, and during this time not much else happens. Also, parts of the instructions might be forgotten or misinterpreted, leading to even more time loss. In such cases, the model reverts to even more general strategies, such as retrieving past experiences from memory:

Decide-retrieve-memory-rule

IF you have to make a certain decision in the current goal (goal buffer)

THEN try to recall an experience that is similar to your current goal (retrieval buffer)

Decide-on-experience-rule

IF you have to make a certain decision in the current goal (goal buffer)

AND you have retrieved a similar experience that went well (retrieval buffer)

THEN make the same decision for the current goal (goal buffer)

This experience-based retrieval strategy retrieves the experience with the highest activation from declarative memory and is based on the assumption that experiences with a high activation are potentially the most relevant in the current situation.

The transition from novice to expert is modeled by ACT-R's mechanism for learning new rules, production compilation (Taatgen & Anderson, 2002). This mechanism takes two existing rules that have been used in sequence and combines them into one rule, given that there are no buffer conflicts (for example, as would be the case when both rules specify using the right hand). An exception is requests to declarative memory: If the first rule requests a fact from declarative memory, and the second rule uses it, the retrieved fact is instead substituted into the new rule. This substitution procedure is the key to learning task-specific rules. For example, the two rules that retrieve an instruction and push a key, together with the instruction to press "enter" when the arrow points to the right plane during landing, would produce the following rule:

IF the goal is to land a plane and your arrow points to the right plane (goal buffer)
AND the right hand is available (manual buffer)
THEN note that the instruction is carried out (goal buffer)
AND push enter (manual buffer)

A rule that retrieves and uses old experiences can also be the source for production compilation. For example, in a situation in which the plane to be landed is a DC10 and the runway is dry, and a previous example in which such a landing was successful on the short runway is retrieved, the following rule would be produced:

IF you have to decide on a runway and the plane is a DC10 and the runway is dry (goal buffer)
THEN decide to take the short runway (goal buffer)

New rules have to "prove themselves" by competing with the parent rule, but once they are established they can be the source for even faster rules. Eventually the model will acquire a rule set that performs like an expert. Comparisons with data from experiments by Ackerman (1988; see Taatgen & Lee, 2003) show that the model predicts the overall performance increase (in terms of number of planes landed) and the individual sub-tasks (e.g., how much time is taken to land a single plane) very well. The model also does reasonably well at the level of individual keystrokes. As an illustration, Figure 25-3 shows the actual and predicted time to land a plane for trials 1 to 10.

An advantage of a model like that of the air traffic control task is that it can serve as a basis for a more general test bed for interface testing. Task-specific knowledge is entered into the system as declarative knowledge, which is very close in form to the instructions provided to the learner. The model can consequently be used to study initial performance and the learning process. Accurate models of human performance also serve as a basis for more advanced forms of individualized user models, which we will discuss in the next section.

Personalized Models

Predictive models can provide helpful indications of how most people will approach and perform tasks. However, both the efficiency with which information can be accessed from the worldwide web and the satisfaction of the user in doing so can be enhanced by adapting websites to individual users, taking into account their different preferences, knowledge, and goals (see Chapter 17 in this volume for more discussion of adaptive and intelligent interfaces). In this section, we describe techniques for determining user characteristics and the need for adaptation, the types of modifications that might be made to adapt websites to individual users, and methods for carrying out the adaptations.

A distinction can be made between adaptable and adaptive systems. *Adaptable systems* (Scerbo, 1996) allow the user to configure the system to suit individual needs. For example,

the user may turn the task of checking spelling during typing over to the software. Once a user allocates a task to the computer in such a system, the computer continues to perform the task until the user modifies the allocation. In *adaptive systems,* the system is designed such that it can modify its behavior itself. Such a system can detect the need to take over or relinquish control of certain tasks and can automatically reallocate them. An example would be a system that detects that many spelling errors are made, and automatically invokes on-line spell checking (see Chapter 15 for more discussion of task automation).

Two things are crucial for an adaptive system to work: The existence of a means to adapt the task, and the ability to detect the need for adaptation. Although it is, in principle, possible to adapt tasks to the observer's state (e.g., by detecting that the observer is under stress and needs to be relieved of some tasks), most adaptive systems react on the basis of a user model that gives an indication of the user's current knowledge, interests, or activity.

Determining User Characteristics

The starting point for any user-based adaptation is the user model (Fischer, 2001). Many websites solicit information about the user directly, by means of a questionnaire (e.g., Fink & Kobsa, 2002). Questions are posed about demographic and personal characteristics and this information is used to create a user profile. The profile of a given user can then be matched to user types stored in a database using a number of techniques. Generally, the determination of how or when to adapt to the user is based on grouping users on some set of features, assuming a degree of homogeneity within the group, and then performing the adaptations that should benefit the average user.

Many user-modeling systems use the technique of predicting unknown characteristics of the user by comparing him or her to similar users. This "collaborative" or "clique-based" filtering may operate according to a clustering algorithm, a correlation-based approach, vector-based similarity technique, or according to a Bayesian network (Fink & Kobsa, 2002;

Webb, Pazzani, & Billsus, 2001). For example, in a correlation-based approach, known characteristics of the current user are used to match (as indicated by a correlation coefficient) the user to other users in the system. Predictions of unknown user characteristics are then computed based on a weighted aggregate of the stored users. One problem in using this technique is determining the number of matches to include in the computations of the predictions. Prediction accuracy tends to be a non-linear function of increasing the number of matches (or, "mentors"), with increments in accuracy decreasing or even reversing as the number of matches becomes large (e.g. Shardanand & Maes, 1995). The optimal number of matches will depend on the strengths of the individual correlations.

Although user characteristics are most commonly determined on the basis of answers to a questionnaire and the use of a statistical clustering algorithm to assign users to groups, information or tasks may also be adapted on the basis of what the user is trying to do, or may even be contingent on the current state of the observer. For example, the system may recognize that the user is experiencing difficulty performing a task and that help needs to be given. Research on adaptive systems in general (e.g., in aviation and process control) has focused on gathering information about the user on the basis of physiological measures such as EEG (electro-encephalogram) and cardiovascular measures. However, because these measures are not readily applied in web settings, they will not be discussed here. Instead, we focus on the behavioral measures of user performance, including actions taken by the operator and the manner in which they are made. Actions may range from mouse button clicks to eye movements, and the measures may range from simply registering that an action has occurred to measuring reaction time or more complex patterns of action. The most practical way to gather information about the user in most web usage situations is by examining the actions performed by the user using the keyboard, touch screen, or mouse. Common measurements

are links accessed and time spent viewing a page. On the basis of these measurements, the website attempts to infer the knowledge and interests of the user.

The knowledge the user possesses can be divided into what was known before visiting a website and what was learned from the website itself. For example, if the user has visited a particular page, it can be assumed afterwards that he is familiar with the information presented there. Information about which sites have been visited can thus be used to modify the presentation of links on subsequent pages. Patterns of actions allow other inferences. For instance, if a user repeatedly chooses a certain category of information, it can be inferred that she knows little about that category. Alternatively, rapid scanning through presentations of information can be assumed to reflect previous acquaintance with (or lack of interest in) the presented information.

A number of systems that use various sources of user information, including eye gaze direction, to determine user interests and information needs have been developed. These systems are "attentive" both in attending to what users do and in presenting information that users will want to attend to. The goal of such systems is to track user behavior, model user interests, and anticipate the actions and desires of the user. For example, the *Simple User Interest Tracker* (*Suitor;* Maglio, Campbell, Barret, & Selker, 2001) analyzes gaze direction and browsing behavior to determine the interests of a user (e.g., which headlines in a news window receive the most interest) and then provides information that should be of interest to the user in a timely manner. The new information is displayed in a peripheral display so that the user can decide whether or not to use the information without being distracted by sudden adaptations.

Updating the User Model on the Basis of Interactions

Many websites can be described as hypermedia environments (settings in which networks of multimedia nodes are connected by links that control information presentation

and image retrieval). Such systems have been in use for a number of years, both on the Internet and off-line. Much of the research on hypermedia environments has focused on educational settings. Proponents of these systems emphasize that such environments enable learner control, thus increasing learner involvement and teaching learners to "learn how to learn" (i.e., to develop metacognitive skills; Large, 1996). However, the randomness with which some learners move through hypermedia environments may limit the effectiveness of the learning (Federico, 1999). User modeling can be an effective tool to increase the efficiency of hypermedia use. A simple example of this related to web use is the tracking of choices in the search for information. Records of the navigational paths followed (so-called "audit trails") may be maintained as users search for information in hypermedia environments. For example, the browser or an independent application might keep track of all pages visited and of the sequences of mouse clicks or menu choices made. It has been argued that audit trails can provide insight into the cognitive and metacognitive processes of the learner that can be used to implement adaptive instructional strategies (Federico, 1999; Milheim & Martin, 1991).

Once an audit trail has been collected, the problem remains of what to do with it. One approach to using this information is to make comparisons across users to determine the search preferences of groups of users. One way of comparing paths is to compute a linearity function. This can be done by finding the ratio of (a) the number of visits to each node in the knowledge network from a parent node to (b) all node visits (Horney, 1993). A hypermedia system that computes linearity functions on-line can then structure information to be compatible with the user's search preferences. Various classifications of users have been made on the basis of navigational paths. On the basis of frequency counts of the number of screens visited, buttons clicked, time spent on each screen, and the depth of search it is possible to distinguish between users who are driven by performance goals (*knowledge*

seekers), who like to find and explore features such as movies (*feature explorers*), who are interested in determining the layout of the site (*cartographers*), and those who are not motivated to learn to use the site (*apathetic users*; Barab, Bowdish, & Lawless, 1997; Lawless & Kulikowich, 1996). Knowledge of the group to which the user belongs can be used to adapt the interface to the user's interests or to give users feedback to enhance their productivity.

Patterns of information acquisition can be used in different ways to infer and adapt to user strategies. For example, a given user may wish to quickly get an overview of a site, and then return to objects that were found especially interesting. Such a user should be provided with a bookmark or annotation tool that facilitates the return to visited objects. Deviations in search type might also signal a need for intervention. For example, a user who begins a session by searching for a target (as indicated by increasing depth in a hierarchy), and who then starts making erratic jumps outside of the hierarchy, could be assumed to be lost and in need of guidance. Much time can be wasted on fruitless searches or in trying to find one's way back to a previous point. A web application that can detect that a user is lost and offer a way back (e.g., by presenting a list of recently visited sites or the front-end of a search engine) might increase search efficiency. Given that many users underutilize history lists and other navigation functions in web browsers, such a facility might be needed (Cockburn & Jones, 1996).

Adaptive Elements

The design of an adaptive system depends on the existence of aspects of the task that can be adapted or automated. Table 25-2 presents a summary of the kinds of tasks and information that might be adapted in a website. Choices regarding what to adapt will depend on the sort of information available on a website and the way in which it is to be used. As an illustration, consider a museum website. In addition to a gift shop and information about opening times and exhibition schedules, the website might contain pictures of, and detailed

and specialized information about, the exhibited objects. Such a website could be used for different purposes. It might be used as a knowledge base, giving access to specific information for research purposes. It might also be used to provide a virtual tour, allowing users who are unable to physically visit the museum to view the objects. Users who have visited the museum, and who want to look up background information about objects they found interesting might also visit it. Although the same database may contain all of the information needed to accommodate these different groups of users, it will have to be accessed and presented in different ways. Thus, the success of the website will depend on the extent to which the wishes of the visitor can be detected and the information can be adapted.

The most obvious adaptation is to tailor the *presentation of information*. In our museum example, presenting all of the information pertaining to a certain object could be overwhelming and impractical. Casual visitors are not likely to be interested in detailed technical information or references to scientific publications, but probably would want to view a reproduction and might be interested in buying one. On the other hand, a researcher is likely to be more interested in detailed information and references and less interested in the souvenir shop. An adaptive interface might note the initial choices made by the user, and adapt subsequent displays to highlight the most likely to be sought after information.

One might also adapt the format of the information presented to the user. Some users will prefer to view pictures of the objects, whereas others will only want to read background information. More generally, the type of question asked by the user may give the website information about how to best present the requested information. For example, “Who was Monet?” is obviously a bibliographic question, and should be answered with bibliographic information. However, the query system may also be able to detect the generality of the question, and could safely assume that the viewer is not familiar with the painting style of the

artist and thus also present representative works (Smeulders, Hardman, Schreiber, & Geusebroek, 2002).

Another aspect of adaptive automation is *task support*. This might involve the scheduling of tasks or automating the execution of tasks. This type of adaptivity is not yet common in web applications because most websites are used for performing just one task at a time. However, on the basis of user search preferences, some tasks might automatically be executed and the results presented to the user. For example, if a website detects that a visitor at the museum site repeatedly returns to a particular object, it might make a quick search of the gift shop and present any object-related products that are available.

The *sequence and manner in which information is entered* can also be adapted to the task or user. Fill-in forms are ubiquitous on the web. By adapting the forms to the user, the filling in of these forms can become less tedious. A simple example is remembering details for specific users so that they need not be repeatedly entered.

The final element that might be adapted is the *choices available to the user*. Nearly every website uses a standard technique to direct the user's attention to specific choices—that of changing the color of recently used links. As discussed in the applications section, below, choices may be informed or modified in a number of additional ways, such as by restricting the availability of links or by offering new links (e.g., informing a user who chooses to view a particular work of art which other works were chosen by people who enjoyed that one).

Another simple adaptation, "smart menus," has been incorporated into many programs. Smart menus reduce search times for finding a desired item and reduce movement time to select the item (Jameson, 2002) by displaying only those menu commands that have been used most often and most recently. Because only the few options that the user is likely to consider are displayed, the need to search the full menu and move the cursor a long distance is eliminated. If the user wants to see the remaining commands, he must indicate that he wants

them to appear by positioning the cursor on an "expand" cue to cause the full menu to appear so that the command can be searched for in the usual manner. Of course, the use of smart menus is to some degree limited by the inconsistency of the mapping of menu items to locations.

Applications

In this section we describe a range of applications that incorporate user models. The applications vary in their degree of adaptability and in the dynamism of the model.

Intelligent Tutors

The basic tenet of intelligent tutors is that information about the user can be used to modify the presentation of information so that learning proceeds more efficiently. Intelligent tutors guide learning and adapt navigational paths to facilitate learner control of knowledge acquisition. A good human tutor could be considered intelligent in this sense if he or she adapted instructional material to the needs of the pupil. For example, a pupil asked to explain a concept might reveal deficiencies of knowledge that the tutor can remediate by means of new examples or repeated explanation. The better the tutor understands the pupil, the better the remediation. In the realm of human-computer interaction, the intelligent tutor needs to possess a user model. Having knowledge of the user is, of course, not enough. The tutor must also have at hand an arsenal of strategies for tailoring the information to the student.

Early attempts to adapt user interfaces to the user's level of expertise were fairly simple in concept, but nonetheless showed the effectiveness of adapting the interface. An example is Carroll and Carrithers's (1984) "training wheels" word processor interface. Carroll and Carrithers found that novice users of a word processor made many mistakes in trying to learn to perform simple tasks, and that most of the mistakes were the result of attempting to carry out unneeded, more advanced commands. They created the training wheels interface by disabling unneeded menu choices, commands, and function keys in order to block errors.

Participants who used the training wheels interface learned to complete simple tasks more quickly than those using the unmodified word processor, and performed better on a comprehension post-test. The major advantage of the simplified interface seemed to be that users spent more time performing criterion tasks and less time recovering from errors. This early research demonstrated that the "learning by discovery" that people tend to engage in can be relatively inefficient because many of the options that are tried out are inappropriate and result in time being devoted to correcting mistakes rather than to learning basic functions.

In the training wheels application, the user model is very simple and not adaptive. Novice users were simply provided with a restricted interface. Thus, training wheels was not "intelligent" in that it did not infer anything about the user and did not adapt itself to user needs. In general, modern word processors are not any smarter, although they do sometimes include proactive help functions. For example, the *office assistant* in Microsoft Word™ pops up from time to time to offer help in completing tasks begun by the user. Unfortunately, it only detects the task the user is attempting to perform, and not whether the user is likely to need help in performing it.

Rather than simply providing hints or eliminating certain options, many tutors attempt to ascertain what the user knows, what they are trying to accomplish, and what specific help the user might need. These intelligent tutors seek to ascertain a student model and adapt the presentation of information to the student based on the current student model. *Student modeling* refers to the techniques and reasoning strategies incorporated in an instructional system to allow it to maintain a current understanding of the student and her activity on the system. The student model can then be used to adapt the learning system to the user by, for example, creating problem sets that reflect the interests and weaknesses (or strengths) of the student, or phasing out assistance in finding solutions to problems at the appropriate time (Federico, 1999).

A distinction can be made between local modeling, which refers to system capabilities for carrying out on-line performance monitoring and error recognition (creating a "trace" of student performance), and global modeling, in which trace data serves as input to an inference engine that maintains a "permanent," evolving view of the student's domain knowledge (Hawkes & Derry, 1996). In most intelligent tutoring systems, local modeling forms the foundation for both assessment of the student and of the tutoring strategy because both of these depend on inferences from the primitive elements detected by the local modeler. The key to effective systems is, then, to develop appropriate local modeling techniques.

Probably the most successful and influential approach to student modeling is the *model-tracing method* (e.g., Anderson, Boyle, & Reiser, 1985). In this technique, knowledge is represented in terms of productions. A "generic" student model contains all the productions an expert problem solver would execute within the problem domain and may also contain a library of "buggy" productions that embody common student mistakes. The expert model and buggy productions form the knowledge base used by the tutor to diagnose student performance. Each step taken by the student is compared to the output of the productions that the generic model would fire under the same circumstances in order to distinguish between valid procedures and actions based on errors or misconceptions. Other student-modeling techniques include using imprecise pattern-matching algorithms that operate on separate semantic and structural knowledge representations (e.g., Hawkes & Derry, 1996) and Bayesian models, which will be discussed next.

Bayesian models are user models based on the Bayesian methods widely used in classifier systems and speech and image recognition. Bayes's theorem states that the probability that hypothesis A is true given that evidence B is observed, is equal to the product of the probability of B being observed given that A is true and the probability of A being true, divided by the sum of the products of the probabilities that B is observed given that

alternative hypotheses are true and the probabilities of these alternative hypotheses. An example of a system that incorporates a Bayesian model is ANDES (Conati, Gertner, & VanLehn, 2002), a tutoring system for helping students learn Newtonian physics. ANDES models the process of mastering the laws of physics and learning how to apply these laws to various problems.

Students start to learn by applying a law (e.g., Newton's second law that mass times acceleration equals net force) in a specific problem context (e.g., the acceleration of a car given that its engine generates a certain amount of force). ANDES observes that the student chooses a specific formula and computes the probability of that action being performed given that various knowledge has been mastered. Using Bayes's theorem, the probabilities in the model, and the observed student actions, the probability that the user has mastered a certain piece of knowledge given his observed action can be calculated. That is, the observed user actions are taken as the evidence, whereas mastery of each piece of knowledge is taken as the various hypotheses.

In most cases, students will need to work examples from different contexts before they are able to apply a law in a consistently correct manner. ANDES captures this with a mechanism that enables the distinction between context-specific and general rules. Within Bayesian modeling, mastery of the general rule is taken as the hypothesis, while mastery of the context-specific rule is used as the evidence. As students solve problems, they gain experience with general laws, increasing their knowledge. The probability that a law has been mastered can be used to deliver specific help. For example, if a student is unable to select a correct formula, but has a high probability of mastering the context-specific rule, he might be given a hint (e.g., that a block resting on a table for which force must be calculated does not move, and therefore has an acceleration of zero). However, if this probability is low, he might be given a more direct hint, such as which formula to apply.

Facilitating Interaction Among Users

User models are increasingly used to facilitate interpersonal information acquisition and retrieval. For example, Harvey, Smith, and Lund's (1998), *InfoVine* uses user models to facilitate information retrieval by relating people to areas of interest or expertise. Bull and McCalla (2002) have developed an agent-based system, *I-Help*, to directly facilitate communication among learners. Using I-Help, learners can participate in private (one-on-one interaction with a partner) and public (entire group) discussions. Each user has their own agent that acts on their behalf. The agent constructs a model of its user's knowledge, eagerness to participate, helpfulness, cognitive style, learning goals, and preferences in a learning partner. This model is based on preferences expressed by the user, peer assessment, and user actions. The agent also constructs partial models of all other users with whom it comes in contact, resulting in just-in-time, distributed, and fragmented user models. When I-Help is invoked to find a partner for discussion, it surveys the characteristics of users who have knowledge about the topic, and presents the user with a ranked list of possible discussion partners. Other functions served by the user models are: protecting users from interruptions, retrieving relevant information for a user, and providing information about a user's current state of knowledge.

Electronic Books

Whether one is writing a handbook chapter or a hypermedia textbook, the aim is usually to structure the instructional material so that a sequence of educational objectives is achieved. The writer uses his expert knowledge to structure the material in order to define an optimal learning path for the average learner. In addition to providing structure by means of sections and section headings, the author may provide navigational or learning tools such as tables of contents, indexes, and glossaries to help the reader in their quest for knowledge. In most books, users can move around freely within the pages but can do little to adapt the book

itself to their own prior knowledge, goals, or learning rates. Hypermedia allows more flexibility in the presentation of material and can potentially be adapted to the user. With the addition of navigation support to adapt hypermedia to the user's learning needs without limiting the free-browsing, learner-controlled nature of the media, the Internet can become an increasingly powerful tool.

One approach to dealing with the enormous amount of text available on the web is to simply print out the documents of interest and to read them the old-fashioned way, on paper. Many authors provide *pdf* files for just this purpose. However, electronic texts maintain several advantages over printed texts, including search functionality and adaptability. In order to use the web effectively, authors must adapt their text to the medium. One such authoring and delivery tool for adaptive electronic texts is *Interbook* (Brusilovsky, Schwarz, & Weber, 1996). This authoring environment uses history-based (annotation based on search patterns), knowledge-based (annotation based on the relations among concepts), and prerequisite-based (annotation which reflects prerequisite knowledge) adaptive annotation of links to guide individual users along an appropriate path through the text. As a reader moves through the hypermedia environment, the content or appearance of the information will depend on the current state of the user model embedded in the hypermedia environment. Dynamic user-model-driven annotation can provide navigation support by, for example, outlining the links to previously visited nodes. For example, Interbook annotates "recommended" topics with a green bullet, "not ready to be learned" topics with a red bullet, and "nothing new" with a white bullet.

As with intelligent tutors, adaptive navigational support depends on a knowledge-based domain model and a user model. In its simplest form, the domain model is a set of concepts (topics, attributes, knowledge elements, objects, learning outcomes, etc.). The domain model may also take the form of a network with nodes corresponding to domain

concepts and links reflecting relationships between concepts. Hypermedia systems generally have a network structure created by indexing hypermedia nodes containing various units of learning material (e.g., presentations, tests, examples, or problems) with domain model concepts. Indexing may be either content based or prerequisite based. With content-based indexing, all concepts related to knowledge contained on a given page are included in the page index (e.g., Schwarz, Brusilovsky, & Weber, 1996) and with prerequisite-based indexing, all concepts that are prerequisite to understanding the content of the page are indexed. For each domain-model concept, an individual user knowledge model stores some value, which is an estimation of the user knowledge level of this concept. This so-called "overlay model" (Brusilosky, 1994) is used to measure independently the user's knowledge of different topics. In Interbook, the user model itself is initialized from a stereotype model, and is modified as the user moves through the information space. Future versions of Interbook should feature an "interview" to further specify the user model and embedded testing for knowledge-based navigation support.

Interbook is currently a focus of research at Carnegie-Mellon University. Basic questions remain to be answered about how and why people use annotations. As the authors put it, "Just as the evaluation of the tool cannot be separated from its development, measuring the effectiveness of its components cannot be successfully achieved in isolation from broader human factors" (Eklund, Brusilovsky, & Schwarz, 1999).

Agents

One purpose of ascertaining user models is to use the models to build *agents* that then operate independently in computing environments. In general, agents are actors in specific environments (such as a network, operating system, or database) that may act in the place of another. *Intelligent agents*, by definition, act independently in interpreting and affecting the environment (Franklin & Graesser, 1996). They may interact with the user and can assist him

by (a) performing tasks on the user's behalf, (b) advising the user on how to perform a task, (c) training or teaching the user, (d) monitoring events or procedures for the user, or (e) helping different users collaborate (see Table 25-4; Tecuci & Hieb, 1996). Information agent technology offers much promise as a key technology for the Internet and worldwide web. A complete inventory of the architectures and implementation of information agents is, however, beyond the scope of this chapter (see Chapter 15 for a more complete discussion of agents). We limit our discussion to several applications and domains in which intelligent agents are being developed.

Information agents are computational software entities that have access to one or more heterogeneous, distributed information sources. They pro-actively search for and maintain relevant information on behalf of their human users or other agents in a dynamic, preferably just-in-time, manner. They are hailed as the solution to managing and overcoming the difficulties associated with information overload in the exponentially growing worldwide web. Advanced information systems increasingly need to balance the autonomy of networked data and knowledge sources with the appropriate use of intelligent information agents (Hendler, 1999).

With the proliferation of information on the worldwide web, interest is focusing on the use of intelligent agents to gather information. Hendler (1999) describes his vision of intelligent information agents. Such agents would accept a commission (e.g., find journal articles describing experiments in a particular field), and determine the relevant constraints (e.g., that a certain paradigm was used) and any preferences (e.g., that the articles were written by research groups at major universities). The internet agent would then find the possibilities and present them for approval. Upon approval, the agent would arrange for delivery of the articles, making any necessary arrangements for payment or registration. As essentially every encounter with a search engine makes clear, even the first function of an

Internet agent—locating relevant information—awaits fulfillment. One problem is that communication between the user and the agent is less than optimal. A good agent must be able to understand the user's goals, preferences, and constraints, and this understanding requires shared knowledge. The agent must therefore be familiar with the vocabulary of the problem area; it must have some domain knowledge to be able to weed out irrelevant information.

Hendler (1999) argues that the key limiting factor in agent technology is the difficulty of building and maintaining ontologies for web use. An ontology is a body of knowledge used in building agents (ontologies are discussed in more detail in Chapter 10). It is often structural in that it consists of a taxonomy of class and subclass relations coupled with definitions of the relationships between them. Ontologies also contain inference rules for classify or linking items, providing the basis for manipulating the terms used in the ontology to allow for a form of communication between user and computer. On this basis, software products that can represent the needs, preferences, and constraints of the user can be created. To be truly useful, an agent (like a graduate student) must have much knowledge of the problem being solved. Building ontologies requires extremely detailed knowledge in a machine-readable form (using artificial intelligence knowledge representation techniques). Market forces are beginning to drive online journals and other scientific content providers to enable greater use of agent-based systems. Current keyword-based search engines are being supplemented by advanced web languages that can organize scientific material, making it accessible to web agents in order to increase agent capability.

In order to act on behalf of the user, an agent must have a good understanding of the needs, wants, and desires of the user. Basically, the agent must know what the user knows or must know even more than the user. As in all knowledge-based systems, knowledge acquisition often forms a bottleneck in the development of agents. One approach to the

problem of building knowledge into agents is the *Disciple* approach of Tecuci and colleagues (e.g., Tecuci, 1992; Tecuci & Hieb, 1996). In the Disciple approach, the user "teaches" the agent by providing specific examples of problems and solutions, explaining these solutions, or supervising the agent as it solves new problems. During these interactions, the agent learns general rules and concepts and expands its knowledge base. The first stage of building a disciple-based agent is *knowledge elicitation*, possibly aided by a knowledge engineer. In this stage the task is thoroughly described in terms of concepts and correlations between them. In the second phase, *apprenticeship learning*, the agent interactively learns from the user on the basis of examples (and solutions) presented by the user, and then experiments by presenting its solutions to the user for verification. It may also request (or require) additional information during this phase. In the third, *autonomous learning* phase, the agent can solve problems without assistance and continues to accumulate experience and exceptions (situations in which correct solutions cannot be found). Finally, *periodic retraining* is required to eliminate exceptions. The Disciple approach has been applied in a variety of domains including manufacturing, instruction, and military command. Its authors argue that the technique is relatively efficient because of the way in which it employs user interpretation.

Agents as Team Members

Simulations are commonly used for training skills that are needed in dangerous or complex environments and for training emergency skills. Recent work (e.g., Dobson et al., 2001) has explored the possibility of using agents to take the role of humans in teams. Team training can be especially challenging because of the need to assemble a number of individuals in order to administer the training. When the training involves experienced or expert individuals, it can be even more difficult to bring the relevant team members together because they cannot be missed in their regular functions. Another challenge of team training

is finding the space to conduct the training. For example, training of emergency procedures in control rooms is made more difficult because the control rooms are always in use for regular monitoring activities. Such considerations can justify the costs of building a simulator for training purposes. The use of virtual reality to build the simulators can enhance their efficiency because the same basic set-up can be adapted to different scenarios. Substituting intelligent agents for some team members can solve the problem of assembling a complete team.

An intelligent agent has the advantage that the level of expertise of this "virtual" team member can be manipulated to meet the training requirements. Specific competencies (or deficiencies) can simply be programmed into its performance. The development of an intelligent agent for team performance requires a working model of the team member being simulated and an understanding of how team members interact with each other. Dobson et al. (2001) found that the introduction of an agent in a training simulation was well accepted by team members. The training was reported as beneficial for illustrating the complex coordination skills required by the team leader and led to increased continuity and confidence within the team.

Social aspects of interaction with agents are of interest both to those building applications, such as team trainers, and to social interaction theorists. Systems with multiple agents can be used to test theories of social interaction or to increase the efficiency of an application (see Brent & Thompson, 1999, for a review). Like people in a social situation, agents in a multi-agent system each have (1) incomplete information or capabilities for solving the problem at hand, (2) no system of global control, (3) decentralized data, and (4) asynchronous computation. As such, multi-agent systems are natural venues for testing theories of social interaction and bounded rationality.

The desirability of incorporating different characteristics of people in an agent depends on the application. Some agents are created with the goal of simulating people, as in the case of the simulated team player or in agents created to act as mentors in a training process (e.g., www.extempo.com), and these agents should obey basic rules of social interaction. Some of the most convincing agents are found in games and advertising contexts. For example, Extempo created a special pet dog "Virtual Jack" to answer questions, share pet facts, explain services, and provide navigation assistance at a commercial pet store site. Customers apparently found the virtual dog entertaining enough to spend time giving it user profile information.

Even when agents are not specifically created to have human-like interaction capabilities, people tend to conceive of computers as if they were people (http://www.Leland.Stanford.edu/group/comdept/). This suggests that frustration could arise when the interactions do not follow established patterns. Agents may also be given emotions, whether to increase the realism of interactions with agents or to model emotional processes (see Picard, 1997, for a review). Other programs use emotion sensing (e.g., sensing of the user's affective state via digitized images of facial expressions or vocal intonation) to adapt interaction to the state of the user (e.g., Essa & Pentland, 1997; Murray & Arnott, 1993).

User Modeling Servers

When developing a user model, a choice has to be made between centralized versus decentralized modeling (Fink & Kobsa, 2000). In decentralized modeling, the user model is a part of the application that is being adapted, and in centralized modeling it is a separate application that interacts with other applications via a client-server framework. Centralized models, sometimes called *user-modeling servers*, offer a number of advantages. First, because information about the user is stored in a central repository, it can be used by many applications, and information gathered during the use of one application can be made

available to different applications. This results in more consistent and complete user models because the adaptations to each application are based on the same models and information. Centralized models are also efficient in that they reduce the need for redundant storage. Furthermore, a centralized user-modeling server makes it much easier to improve and extend user models since any changes will automatically be available to all applications. Finally, a single user-modeling server has advantages in security. Since all information is in one location, it is much easier to guarantee security and privacy. Of course, centralized modeling has the disadvantage that it introduces a single point of failure: If the user-modeling server goes down, all user modeling goes with it. Good network connections and powerful hardware resources are necessary for these systems to work. Despite this potential problem, Fink and Kobsa argue that user-modeling servers are especially suitable for websites that deal with a large number of visitors having a wide range of interests that can only be answered by combining different sources of information.

An overview of commercial user-modeling servers can be found in Fink and Kobsa (2000). Most systems are intended for company-wide use, combining information from various departments to create user models that are as complete as possible. User-modeling servers gather their information by communication with various applications. In addition to passively receiving information, some systems can actively request certain information from the applications. User-modeling servers contain reasoning mechanisms that allow them to make classifications of users, recommendations for users, or predictions of user behavior that can be communicated back to the applications. For example, information about past purchases and pages requested, as well as financial information about the customer, can be combined for use throughout a company.

Adapting to Physical Limitations

Although the focus of this chapter is on acquiring knowledge and making inferences about cognitive strategies and adaptations based on them, other aspects of human-machine interaction may be also be improved by user modeling. For example, Trewin and Pain (1999) have demonstrated how personalized user modeling of keyboard operation can be used to dynamically adapt keyboard operation to people with various motor disabilities. Their model follows the keystrokes made by the user and detects difficulties caused by (1) holding keys depressed for too long, (2) accidentally hitting keys other than the intended one, (3) unintentionally pressing a key more than once, or (4) problems in using the "shift" or other modifier keys. For example, if the model observes that a typist uses the "caps lock" key to type just one capital letter, or presses and releases the "shift" key without modifying a key, or presses and releases the shift key more than once without typing any letter, it will invoke the "sticky keys" facility which causes the shift key to stay active after having been pressed until the next alphanumeric character is pressed. Trewin and Pain showed that the model performed well in differentiating between people who needed interventions to improve their typing and those who did not. Because it was able to make recommendations after just 20 keypresses, the model may be of practical use in situations (such as libraries or internet cafes) where users use public terminals.

Summary

Essentially every application of user modeling represents an effort to move beyond direct manipulation of computer software via commands entered using a keyboard, mouse, or touch screen to adaptive or even autonomous behavior on the part of the software. In other words, they contribute to a situation in which the process of using a computer is cooperative, and both human and computer agents can initiate communication, monitor events and perform tasks to meet a user's goals (Shneiderman & Maes, 1997). Not all researchers are convinced

that such a trend is desirable. For example, Shneiderman (Shneiderman & Maes, 1997) questions whether the computer will ever be able to automatically ascertain the users' intentions or to take action based on vague statements of goals. He also suggests that users will be reluctant to relinquish the control over the system afforded by direct manipulation, and warns that systems that adapt to the user or carry out actions on their behalf may be hard to understand. Despite such concerns, many people have argued that further increases in the usability of software require that the software itself infer some of the actions to be taken.

References

Ackerman, P. L. (1988). Determinants of individual differences during skill acquisition: Cognitive abilities and information processing. *Journal of Experimental Psychology: General, 117*, 288-318.

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Erlbaum.

Anderson, J. R., Boyle, C. F., & Reiser, J. F. (1985). Intelligent tutoring systems. *Science, 228*, 456-468.

Anderson, J. R., Bothell, D., Byrne, M. D., & Lebiere, C. (2002). An integrated theory of the mind. Retrieved October 17, 2002, from http://act-r.psy.cmu.edu/papers/403/IntegratedTheory.pdf

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Erlbaum.

Barab, S., Bowdish, B., & Lawless, K. (1997). Hypermedia navigation: Profiles of hypermedia users. *Educational Technology Research and Development, 45*, 23-41.

Brent, E., & Thompson, G. A. (1999). Sociology: Modeling social interaction with autonomous agents. *Social Science Computer Review, 17*, 313-322.

Brusilovsky, P. (1994). Student model centered architecture for intelligent learning environments. In *Proceedings of the Fourth International Conference on User Modeling* (pp. 31-36), 15-19 August, Hyannis, MA.

Brusilovsky, P. (1996). Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction, 6*, 87-129.

Brusilovsky, P., Schwarz, E., & Weber, G. (1996). A tool for developing adaptive electronic textbooks on WWW. In *Proceedings of WebNet'96, World Conference of the Web Society* (pp. 64-69). San Francisco, CA, October 15-19.

Byrne, M.D. (2001). ACT-R/PM and menu selection: Applying a cognitive architecture to HCI. *International Journal of Human-Computer Studies, 55*, 41-84.

Byrne, M. D. & Anderson, J.R. (2001). Serial modules in parallel: The psychological refractory period and perfect time-sharing. *Psychological Review, 108*, 847-869.

Bull, S., & McCalla, G. (2002). Modelling cognitive style in a peer help network. *Instructional Science, 30*, 497-528.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Erlbaum.

Carroll, J. M., & Carrithers, C. (1984). Blocking learner error states in a training-wheels system. *Human Factors, 26*, 377-389.

Chong, R. S. (1999). Modeling dual-task performance improvement: Casting executive process knowledge acquisition as strategy refinement. Unpublished dissertation. University of Michigan.

Chung, J., & Reigeluth, C. (1992). Instructional prescriptions for learner control. *Educational Technology, 32,* 14-20.

Cockburn, A., & Jones, S. (1996). Which way now? Analyzing and easing inadequacies in WWW navigation. *International Journal of Human-Computer Studies, 45*, 105-129.

Conati C., Gertner, A., & VanLehn, K. (2002) Using Bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction 12*, 371-417.

Daily, L. Z., Lovett, M. V., & Reder, L. M. (2001). Modeling individual differences in working memory performance: A source activation account. *Cognitive Science, 25*, 315-353.

Dobson, M. W., Pengelly, M., Sime, J.-A., Albaladejo, S. A., Garcia, E. V., Gonzales, F., & Maseda, J. M. (2001). Situated learning with co-operative agent simulations in team training. *Computers in Human Behavior, 17*, 547-573.

Eklund, J., Brusilovsky, P., & Schwarz, E. (1999). Adaptive Textbooks on the World Wide Web. *UniServe Science News, 12*.

Essa, I., & Pentland, A. (1997). Coding, analysis, interpretation, and recognition of facial expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 19*, 757-763.

Federico, P.-A. (1999). Hypermedia environments and adaptive instruction. *Computers in Human Behavior, 15*, 653-692.

Fink, J. & Kobsa, A. (2000). A review and analysis of commercial user modelling servers for personalization of the world wide web. *User Modeling and User-Adapted Interaction, 10*, 209-249.

Fink, J., & Kobsa, A. (2002). User modeling for personalized city tours. *Artificial Intelligence Review, 18*, 33-74.

Fischer, G. (2001) User modeling in human-computer interaction. *User Modeling and User-Adapted Interaction, 11*, 65-86.

Franklin, S., & Graesser, A. (1996). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Amsterdam: Springer-Verlag.

Harvey, C. F., Smith, P., & Lund, P. (1998). Providing a networked future for interpersonal information retrieval: InfoVine and user modeling. *Interacting with Computers, 10*, 195-212.

Hawkes, L. W., & Derry, S. J. (1996). Advances in local student modeling using informal fuzzy reasoning. *International Journal of Human-Computer Studies, 45*, 697-722.

Hendler, J. (1999). Is there an intelligent agent in your future? *Nature, Webmatters*, 11 March.

Horney, M. (1993). A measure of hypertext linearity. *Journal of Educational Multimedia and Hypermedia, 2*, 67-82.

Hornof, A.J. & Kieras, D.E. (1997). Cognitive modeling reveals menu search is both random and systematic. In *Proceedings of the CHI '97 conference on human factors in computing systems* (pp. 107-114). New York: ACM Press.

Howes, A. (1994). A model of the acquisition of menu knowledge by exploration. In B. Adelson, S. Dumais & J. R. Olson (Eds.), *Proceedings of CHI '94: Human Factors in Computing Systems* (pp. 445-451). New York: ACM Press.

Jameson, A. (2002). Adaptive interfaces and agents. In J. A. Jacko & A. Sears (Eds.), *The human-computer interaction handbook: Fundamentals, evolving technologies, and emerging applications* (pp. 305-330). Mahwah, NJ: Lawrence Erlbaum.

Kieras, D. E. & Meyer, D. E (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-computer Interaction, 12*, 391-438.

Kitajima, M. & Polson, P.G. (1997). A comprehension-based model of exploration. *Human-computer Interaction, 12*, 345-389.

Lawless, K., & Kulikowich, J. (1996). Understanding hypertext navigation through cluster analysis. *Journal of Educational Computing Research, 14*, 385-399.

Large, A. (1996). Hypertext instructional programs and learner control: A research review. *Education for Information, 4,* 95-106.

Maglio, P., Campbell, C., Barret, R., & Selker, T.(2001) An architecture for developing attentive information systems. *Knowledge-Based Systems, 14,* 1-103.

Milheim, W., & Martin, B. (1991). Theoretical bases for the use of learner control: Three different perspectives. *Journal of Computer-based Instruction, 18*, 99-105.

Murray, I. R., & Arnott, J. L. (1993). Toward the simulation of emotion in synthetic speech: A review of the literature on human vocal emotion. *Journal of the Acoustic Society of America, 93*, 1097-1108.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, M.A.: Harvard University Press.

Picard, R. (1997). *Affective computing*. Cambridge, MA: MIT Press.

Pirolli, P. & Fu, W.T. (2003). SNIF-ACT: A model of information foraging on the world wide web. Proceedings of the Ninth International Conference on User Modeling (pp. 45-54|). Heidelberg, Germany: Springer Verlag.

Reber, A. S. (1989). Implicit learning and tacit knowledge. *Journal of Experimental Psychology, 118*, 219-235.

Salvucci, D. D. & Macuga, K. L. (2001). Predicting the effects of cell-phone dialing on driver performance. In: E. M. Altmann & A. Cleeremans (eds.), Proceedings of the 2001 Fourth International Conference on Cognitive Modeling (pp. 25-30). Mahwah, NJ: Erlbaum

Scerbo, M. W. (1996).Theoretical perspectives on adaptive automation. In R. Parasuraman & M. Mouloua (Eds), *Automation and human performance: Theory and applications.* (pp. 37-63). Hillsdale, NJ: Erlbaum.

Schwarz, E., Brusilovsky, P., & Weber, G. (1996) Worldwide intelligent textbooks. Proceedings of ED-TELECOM ’96 - World Conference on Educational Telecommunications (pp. 302-307). Boston, MA, June 1-22, 1996.

Shardanand, U., & Maes, P. (1995). Social information filtering: Algorithms for automating “word of mouth”. In *Proceedings of CHI-95* (pp. 210-217). Denver, CO.

Shneiderman, B., & Maes, P. (1997). Direct manipulation vs. interface agents. *Interactions, 4,* 42-61.

Smeulders, A. W. M., Hardman, L., Schreiber, G., & Geusebroek, J. M. (2002). An integrated multimedia approach to cultural heritage e-documents. ACM workshop on multimedia information retrieval (http://www.science.uva.nl/~mark/pub/2002/smeulderrsMIR02.pdf).

Taatgen, N. A. (2002). A model of individual differences in skill acquisition in the Kanfer-Ackerman Air Traffic Control Task. *Cognitive Systems Research, 3*, 103-112.

Taatgen, N. A., & Anderson, J. R. (2002). Why do children learn to say "Broke"? A model of the past tense without feedback. *Cognition, 86*, 123-155.

Taatgen, N. A., & Lee, F. J. (2003). Production Compilation: A simple mechanism to model complex skill acquisition. *Human Factors, 45,* 61-76.

Tecuci, G. (1992). Automating knowledge acquisition as extending, updating and improving a knowledge base. *IEEE Transactions of Systems, Man, and Cybernetics, 22,* 1444-1460.

Tecuci, G., & Hieb, M. R. (1996). Teaching intelligent agents: The Disciple approach. *International Journal of Human-Computer Interaction, 8*, 259-285.

Trewin, S., & Pain, H. (1999). A model of keyboard configuration requirements. *Behaviour & Information Technology, 18*, 27-35.

Webb, G., Pazzani, M., & Billsus, D. (2001) Machine learning for user modeling. *User Modeling and User-Adapted Interaction, 11,* 19-29

Young, R. M. & Lewis, R. L. (1999). The Soar Cognitive Architecture and Human Working Memory. In: A. Miyake & P. Shah (Eds.), *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control* (pp. 224-256). Cambridge: Cambridge University Press.

Table 25-1. Overview of Constraints that the Model Human Processor (MHP), Soar, EPIC and ACT-R Impose on Cognitive Processing

Process	Model	Constraint	Reference
Working Memory			
	MHP	Working memory has a limited capacity of 5-9 chunks and decays in 900-3500 ms.	Card, Moran, & Newell (1983)
	Soar	Limitations of working memory arise on functional grounds, usually due to lack of reasoning procedures to properly process information.	Young & Lewis (1999)
	ACT-R	Limitations of working memory arise from decay and interference in declarative memory. Individual differences are explained by differences in spreading activation.	Daily, Lovett, & Reder (2001)
Cognitive performance			
	MHP	The cognitive processor performs one recognize-act cycle every 25-170 ms, in which the contents of working memory initiate actions that are linked to them in long-term memory.	Card, Moran, & Newell (1983)
	Soar	A decision cycle in Soar takes 50 ms,	Newell (1990)

	multiple productions may fire but only when leading to a single choice.	
ACT-R	A production rule takes 50 ms to fire, no parallel firing is allowed. A rule is limited to inspecting the current contents of the perceptual and memory-retrieval systems and initiating motor action and memory-retrieval requests.	Anderson, Bothell, Byrne, & Lebiere (2002)
EPIC	Production rules take 50 ms to fire, but parallel firing of rules is allowed.	Kieras & Meyer (1997)
Perceptual and Motor systems		
MHP	Perceptual processor takes 50-200 ms to process information, motor processor 30-100 ms. Duration of motor actions is determined by Fitts's Law.	Card, Moran, & Newell (1983)
EPIC	Perceptual and motor modules are based on timing from the MHP. Modules operate asynchronously alongside central cognition.	Kieras & Meyer (1997)
ACT-R; Soar	Both use modules adapted from EPIC.	Byrne & Anderson (2001), Chong (1999)
Learning		
MHP	Speed up in performance is according to the power law of practice.	Card, Moran, & Newell (1983)

Soar	Learning is keyed to so-called impasses, where a sub-goal is needed to resolve a choice problem in the main goal.	Newell (1990)
ACT-R	Learning is based on rational analysis in which knowledge is added and maintained in memory on the basis of expected use and utility.	Anderson, Bothell, Byrne, & Lebiere (2002)

Table 25-2. Adaptable Aspects of Web Use.

Information Presentation	Selecting the right information
	Presenting information in the best format
Task support	Scheduling of tasks
	Automating tasks
Information Input	Guiding information input
	Checking information input
Adapting choices	Presenting appropriate links
	Coding links

Table 25-3. Functions of a User-modeling System

1. Timely identification of user interests and preferences based on observable behavior.
2. Assignment of users to groups so that subgroup stereotypes can be used to predict interests and preferences.
3. Use of domain knowledge to infer additional interests and preferences.
4. Storage and updating of explicitly provided information and implicitly acquired knowledge.
5. Guarding the consistency and privacy of user models.
6. Provision of user model information to authorized applications.

Figure Captions

Figure 25-1. A simplified version of the MHP comprising a memory system (working memory and long-term memory) and three processors (perceptual, cognitive, and motor).

Figure 25-2. Actual search times (obtained with human subjects) and predictions of the serial search model and the overlapping search model.

Figure 25-3. The actual and predicted time to perform the landing unit task for trials 1 to 10 in the task of Taatgen & Anderson, 2002.

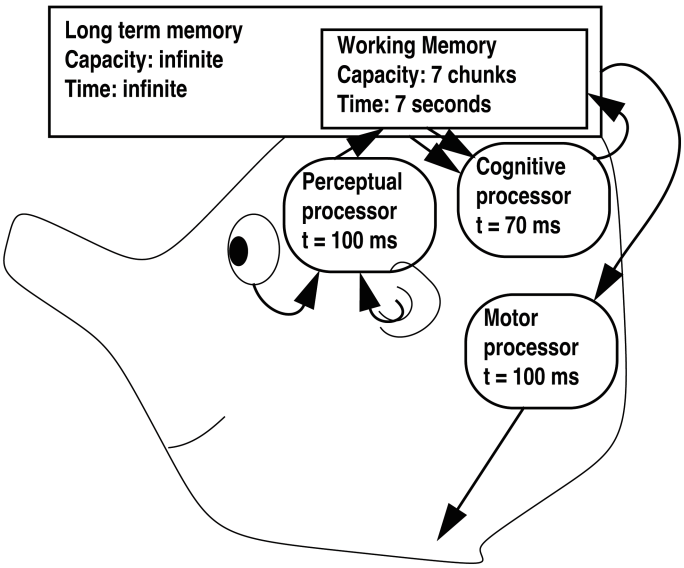
Long term memory
Capacity: infinite
Time: infinite
Working Memory
Capacity: 7 chunks
Time: 7 seconds
Perceptual
processor
t = 100 ms
Cognitive
processor
t = 70 ms
Motor
processor
t = 100 ms

Figure 25-1.

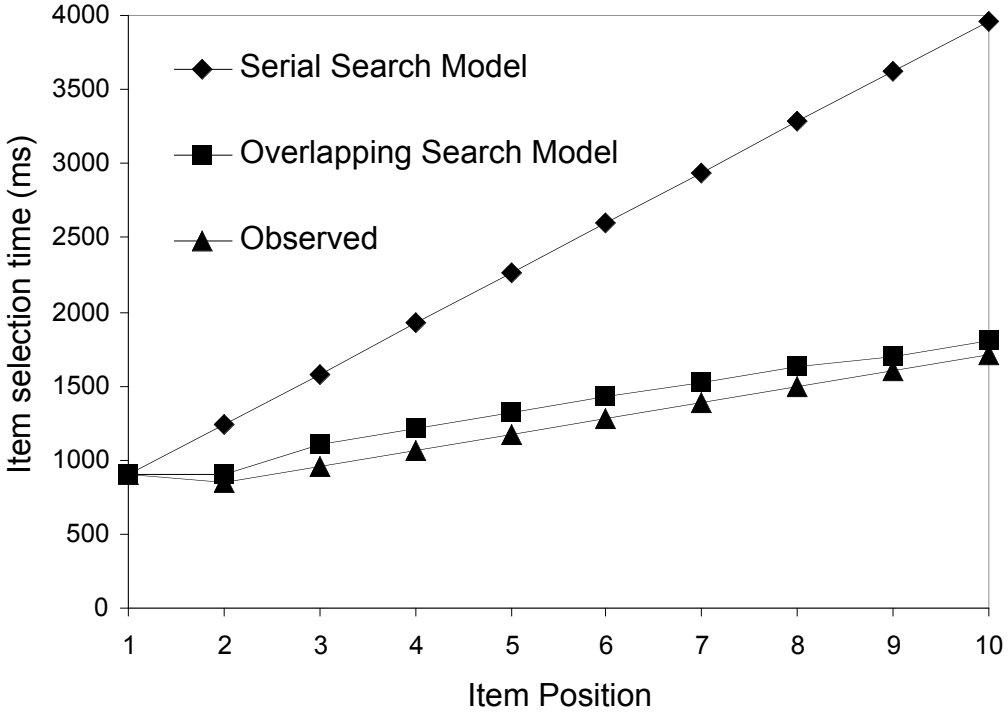
Serial Search Model
Overlapping Search Model
Observed
Item selection time (ms)
Item Position
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
3
4
5
6
7
8
9
10

Figure 25-2

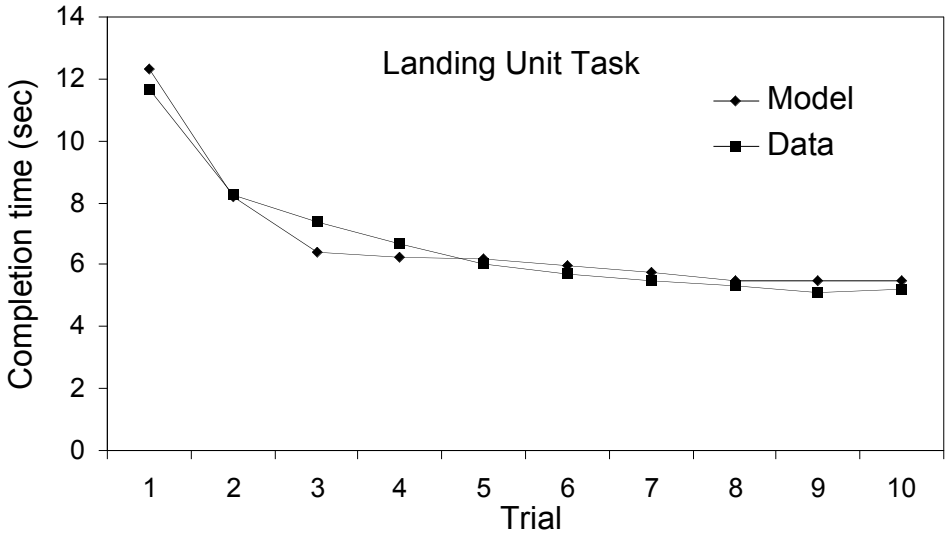
Landing Unit Task
Model
Data
Completion time (sec)
Trial
0
2
4
6
8
10
12
14
1
2
3
4
5
6
7
8
9
10

Figure 25-3