Cognitive Science 29 (2005) 421–455

Modeling Parallelization and Flexibility Improvements in Skill Acquisition: From Dual Tasks to Complex Dynamic Skills

Niels Taatgen

Department of Psychology, Carnegie Mellon University
and
Department of Artificial Intelligence, University of Groningen, Netherlands

Received 30 April 2004; received in revised form 19 November 2004; accepted 2 February 2005

Abstract

Emerging parallel processing and increased flexibility during the acquisition of cognitive skills form a combination that is hard to reconcile with rule-based models that often produce brittle behavior. Rule-based models can exhibit these properties by adhering to 2 principles: that the model gradually learns task-specific rules from instructions and experience, and that bottom-up processing is used whenever possible. In a model of learning perfect time-sharing in dual tasks (Schumacher et al., 2001), speedup learning and bottom-up activation of instructions can explain parallel behavior. In a model of a complex dynamic task (Carnegie Mellon University Aegis Simulation Program [CMU-ASP], Anderson et al., 2004), parallel behavior is explained by the transition from serially organized instructions to rules that are activated by both top-down (goal-driven) and bottom-up (perceptually driven) factors. Parallelism lets the model opportunistically reorder instructions, leading to the gradual emergence of new task strategies.

Keywords: Dual tasking, Psychology, Cognitive architecture, Complex systems, Human–computer interaction, Instruction, Learning, Situated cognition, Skill acquisition and learning, Knowledge representation, Symbolic computational modeling

1. Introduction

Rules and production systems are a popular choice of representation in modeling human cognition and have been widely successful in providing accurate and insightful accounts of human reasoning. However, such models have also been widely criticized. A first criticism is that models based on rules already have all the knowledge needed to solve the problem or do the

Requests for reprints should be sent to Niels Taatgen, Department of Psychology, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213. E-mail: taatgen@cmu.edu

task (e.g., Dreyfus & Dreyfus, 1986; Rumelhart & McClelland, 1986). According to these critics, rule-based models may exhibit human-like performance on tasks, but fail with respect to their ability to offer an explanation of how such knowledge is acquired. For example, in a typical psychology experiment the participants are given instructions. Just on the basis of these instructions people are typically capable of doing the task. It requires quite a leap of faith to assume that instructions are automatically translated into rules. Many problems usually associated with insight problem solving also require the problem solver to go beyond what is stated in instructions, for example, by restructuring the problem. Another aspect of acquiring a task on the basis of instructions, which is hard to reconcile with the assumption that rules are learned directly from them, is that, although people can often perform the task based just on the instructions, they are typically slow initially and make many errors.

A second, related criticism of rule-based models is their "brittleness" (e.g., Holland, 1986): their behavior is limited to what their rules let them do. This may mean that if the model must perform a task that is similar, but not identical, to the original task, it may not work at all. This is an especially problematic issue when a model must operate in the real world, for example, in a robot or an information-foraging agent on the Web where unexpected situations can occur that are not covered by its rules. But brittle models also fall short of accounting for the flexibility and fine details of human behavior. The problem of brittleness has often been used as an argument against rule-based models as a viable approach to modeling human performance. For example, neural network models (e.g., Rumelhart & McClelland, 1986) can structure themselves on the basis of experience and need little knowledge at the outset. They can also adapt themselves surprisingly well to new situations, avoiding the brittleness problem. However, within this framework it is hard to model situations in which there is prior knowledge or a set of instructions to carry out.

We present an alternative view to problems with traditional rule-based models: It is not the rule representation itself that is problematic, but certain assumptions behind many rule-based models. Two assumptions in particular will be challenged in this article. The first is that, when participants in an experiment are given a new task, they almost immediately have access to a set of task-specific rules to perform it (the *instruction assumption*). Although this assumption is quite unreasonable, there are nevertheless many models that make it. As an alternative, following our earlier work (Taatgen & Lee, 2003), we employ models that gradually learn their rules during task performance on the basis of the instructions and background knowledge.

The second assumption of many rule-based models is that task representations are strictly hierarchical (i.e., Card, Moran, & Newell, 1983). The idea behind hierarchical task decomposition is that the task is decomposed into subtasks, each of which can be further decomposed into either deeper subtasks or sequences of primitive steps. Hierarchical task decomposition is based on sound software engineering principles and has been successful in describing task performance at higher levels, but it leads to brittleness at the finer levels of detail. The reason is that a strict hierarchical model enforces pure top-down control. Regardless of the situation in the world, the model follows its plan, even if it no longer makes sense given the context. Although brittleness can become evident in many ways, we focus on a category where it is particularly evident: multitasking situations, which can occur when either two or more tasks have to be done in parallel, or when different steps in a single task can partially or completely overlap

in time (within-task multitasking). Such situations are particularly hard to model from a purely top-down perspective, because it assumes that it is possible to schedule all the components in the subtasks ahead of time. The alternative to top-down control is bottom-up control. Here the input from the environment is in full control. This approach is obviously not viable for all tasks because many require some form of internal state representation. To stress that a model should have as much bottom-up control as possible, we introduce the *minimal control principle,* which posits that humans strive for a strategy with a minimal number of control states. A control state marks where the strategy is in the task execution. In a fully hierarchical task representation every step in the process has its own control state, marking exactly where in the hierarchy the strategy is at that time. The disadvantage is that a fully flexible strategy should be able to address any possible event in any possible control state. The more control states there are, the larger the danger that there is some combination of a control state and an event that the strategy cannot handle, leading to brittle behavior. A strategy with a minimum number of control states produces a combination of top-down and bottom-up control, where the former is only used when needed, which is the case when two different situations cannot be distinguished on the basis of information that is available through bottom-up processes.

The general framework within which we develop these ideas involves the acquisition of skills in dynamic tasks in which timing and interaction with the outside world play an important role. This is a suitable domain for criticizing some conventional assumptions underlying rule systems, as well as for testing the adequacy of our own rule-based models. Skill acquisition has always been an important topic in cognitive modeling research, but the focus has been mainly on explaining speed improvements (e.g., Anderson, 1982; Logan, 1988; Newell & Rosenbloom, 1981; Taatgen & Lee, 2003). Another characteristic of a skill at the expert level that usually is not modeled is the reduced demand on attention. The classical example is that of a skilled driver having a conversation with his or her passenger, which is very hard for a novice driver. Various researchers have constructed cognitive models that can explain this ability to do things in parallel (Byrne & Anderson, 2001b; Freed, Matessa, Remington, & Vera, 2003; Gray, John, & Atwood, 1993; Meyer & Kieras, 1997), but these are expert models that do not learn and therefore rely, in some sense, on the instruction assumption. All of these models agree on the fact that parallel behavior can be achieved by the appropriate parallelization and interleaving of cognitive, perceptual, and motor actions, although there is still debate on whether cognitive actions can be carried out in parallel.

The goal of this article is to show how models of acquiring dynamic skills from instructions can make a start on the instruction and brittleness problems. The models are outfitted with task-general production rules that can operate instructions stored more or less literally. A relatively slow learning process then transfers the literary representation of the instructions into task-specific rules. We show that models with fewer control states, following the minimal control principle, produce more flexible skills and give a better account of the human data.

We first review some existing models of both skill acquisition and expert-level dual tasking. We then proceed to explain the model of a set of dual-tasking experiments by Schumacher et al. (2001), in which participants achieve perfect time-sharing between two simple tasks, along with the relevant aspects of the Adaptive Control of Thought–Rational (ACT–R) architecture (Anderson et al., 2004). We will see that the model of dual tasking learns a pure bottom-up strategy that translates into a model with just one control state. We then present a second model

of a complex dynamic task, Carnegie Mellon University Aegis Simulation Program (CMU-ASP; Anderson et al, 2004), in which participants classify airplanes on a radar screen. This task has many opportunities for within-task multitasking and the usage of different strategies that can only be brought to bear if the task representation is sufficiently loose to allow exploration of these opportunities. Although this task needs at least some control states, the model presented here drastically reduces the number that would be required by a top-down model.

1.1. Cognitive models of skill acquisition

Skill acquisition involves progressing from slow, deliberate processing to fast, automated processing. Many models of skill acquisition assume that, by performing a task, learning processes transform the initial representation of the task into increasingly efficient representations. One of the first models proposed by Anderson (1982), based on Fitts's (1964) three-stage theory of skill acquisition, assumes that knowledge for a new skill is initially stored as declarative knowledge, using a simple propositional representation. Each declarative element is a symbol with labeled references to other elements. For example, the fact 3 + 4 = 7 is represented by a symbol with references to "addition," "3," "4," and "7." Declarative memory can also be used to store a set of instructions. An example is a declarative instruction for a recipe, such as making tea: Put water in kettle, put water on stove until it boils, put tea leaves in teapot, pour boiling water in teapot, and wait 3 to 5 min. These five instructions for making tea can be stored almost literally in declarative memory. The simplicity of the representation explains why this is the starting point for a new skill: Declarative items of knowledge can be added as single items to memory. The disadvantage of declarative representations is that they cannot act by themselves; instead they need, according to Anderson's theory, production rules to be retrieved from memory and interpreted. This explains why initially processing is slow, because the declarative representations must be retrieved before they can be carried out, and it is prone to errors because the right declarative fact might not be retrieved at the right time. Another reason may be that initial knowledge is insufficient, so explicit reasoning is necessary to fill in the gaps. Finally, if declarative instructions are retrieved one at a time, it is often impossible to notice the potential for multitasking. In the tea recipe example, it would make sense to put the leaves in the teapot while you are waiting for the water to boil. A strict linear interpreter of the instructions would miss such an opportunity. While the model is processing declarative knowledge, several learning mechanisms start to gradually transform the declarative knowledge into production rules. In Anderson's (1982) original ACT* theory this was a set of mechanisms: proceduralization, composition, specialization, and generalization. One disadvantage of this theory was that these mechanisms together produced rules at such a rate that they could not be properly evaluated. This ACT–R is much more tractable because it has only one rule-learning mechanism, compilation, which we will discuss later. Once knowledge is transformed into a procedural representation, it is more efficient to use, because the inefficient retrieval step can be skipped.

Optimizing future efficiency on the basis of experience is the focus of two other learning models, chunking (Newell & Rosenbloom, 1981) and instance theory (Logan, 1988). Chunking, the learning mechanism employed by the SOAR cognitive architecture (Newell,

1990), learns new production rules, not from declarative knowledge, as in Anderson's theory (1982), but from a problem-solving trace that started with an impasse. The general idea is that novices begin with general problem-solving skills and domain knowledge, but not with appropriate rules to solve the problem right away. The novice uses these general skills to solve the problem, leading to search and slow performance. The chunking mechanism learns new production rules on the basis of these problem-solving attempts. These new rules can be used on future occasions where the same or a similar problem is encountered. A related model is provided by Logan's (1988) instance theory, which assumes two competing strategies for solving a problem: a general algorithm and a retrieval process. The algorithm is capable of solving the problem, in principle, but will take substantial time. An example is to solve an alphabet arithmetic equation such as *F* +4 = ? by counting four steps from *F*. Once the algorithm has solved a certain problem, the answer (e.g., *F* + 4 = *J*) is stored in memory. On future occasions the retrieval process tries to retrieve these past answers, called *instances* in Logan's theory. If *F* + 4 =? is presented again, the algorithm and the retrieval process simultaneously try to find the answer, and the strategy that finishes first wins the competition.

Our own previous work (Taatgen & Lee, 2002) concerns skill acquisition in ACT–R and also used the idea of gradually transforming declarative into procedural knowledge. In a model of the Kanfer–Ackermann Air Traffic Controller task (Ackerman, 1988), we demonstrated how an ACT–R model can learn the task from instructions. The model matched the experimental results on a global level (score per trial) and intermediate level (time per unit task), but it could not capture performance at the level of individual keystrokes. The problem was that landing each plane required some planning (determining which plane had to be directed to what runway) and then execution through a series of keystrokes. Participants in the experiments were perfectly capable of interleaving planning and execution, but the model could not, due to its hierarchical representation of the task.

Despite differences in representation and mechanisms, all four models share global similarities. In each case there is a transition from general task-independent methods to knowledge representations that are tailored to the task on the basis of experience. All four models produce a speedup in performance and a reduction in errors, as far as they are modeled, but they differ in whether and how they address other aspects of automaticity.

1.2. Models of parallel behavior

There are several theories on how multiple tasks can be carried out at the same time, but they share the same basic idea that the cognitive system consists of several modules that can each operate asynchronously and independently from each other. An early version of this idea is found in CPM-GOMS (Gray et al., 1993), which assumes that cognitive, perception, and motor actions are done in parallel, although within each modality actions are serial. This idea is also the foundation of the Executive-Process Interactive Control (EPIC; Meyer & Kieras, 1997) architecture, except that EPIC assumes that cognitive steps can be executed in parallel. Parallelization of behavior in these theories is a matter of optimizing the schedule of actions in all these modules. The APEX architecture (Freed et al., 2003) actually uses an algorithm to derive an optimal schedule. The peripheral modules of EPIC have been adopted into the ACT–R architecture (Byrne & Anderson, 2001b), but ACT–R continues to assume that central cognition is serial.

Parallel behavior is often studied in dual-task experimental paradigms. The dominant experimental paradigm is the *psychological refractory period* experiment (e.g., Pashler, 1994), in which participants do two tasks at the same time, but with the instruction to give priority to one task. The typical finding is that performance on the second task is slower than when the second task would have been carried out alone, indicating a cost of having to do both tasks at the same time. Schumacher et al. (2001) argued that this cost of doing two tasks at the same time may be an artifact of the priority given to one task. As a consequence, participants might postpone their response on the second task to make sure it is not made before the response on Task 1. To test this idea they asked participants to do two tasks in parallel with no order restrictions: a visual–manual task, in which a visual stimulus required a certain motor response, and an aural–vocal task, in which an aural stimulus required a certain vocal response. Both stimuli appeared at the same time, and the participant was instructed to react as fast as possible on both tasks. Schumacher et al. found that, given sufficient training, participants achieved perfect time-sharing, enabling them to do both tasks as fast as they would each task separately.

It is important to note that the usefulness of parallel processing is not restricted to multitask contexts. In many single tasks, such as the making-the-tea example, one can do several steps in parallel, and explaining this poses the same sort of questions as multitasking paradigms. Models that are based on parallel asynchronous modules have been quite successful in explaining various dual-task effects, as well as applied tasks such as menu and icon search (e.g., Byrne & Anderson, 2001a; Hornof & Kieras, 1997). The challenge is to explain how these efficient models can be learned.

1.3. Models of learning parallel behavior

Although instance theory does not deal with parallelism directly, it does account for why automated processes require less attention than controlled processes. According to Logan (1992), automation is characterized by diminishing attention requirements because, once attention is focused on a visual stimulus, the encoding and instance retrieval processes follow automatically and do not require attention, as opposed to the general algorithm. Despite the fact that instance theory sheds some light on the issue of attention, it is incomplete in the sense that it does not deal with the control issues that arise in complex skills. Kieras, Meyer, Ballas, and Lauber (2000) have such a theory, which posits five stages that the acquisition of multitasking must traverse, and they illustrated most of these with EPIC models. The stages are derived from analogies with how operating systems schedule multiple processes. In Stage 0, preprocedural interpretative multitasking (which they do not model), tasks are still encoded declaratively and must be retrieved and interpreted to be carried out. Stage 1 involves general hierarchical competitive multitasking, in which a general executive controls which task carries out its actions. While one task is performing its actions, the other tasks are "locked out" and are not allowed to carry out any step. In Stages 2 to 4, scheduling becomes increasingly more customized, eventually leading to a task-specific scheduler that is tailored to the specific task and that interleaves steps optimally. Although Kieras et al. specified the stages of learning, they did not provide a learning mechanism to accomplish it. Note that some of the problems to be solved in the EPIC model are unique to its hypothesis that central cognition is parallel, such as the need to "lock out" certain tasks. This implies that parallelism must be prevented early in the acquisition of a skill, as opposed to being learned later.

Chong and Laird (1997) modeled some aspects of learning dual tasking in their EPIC–SOAR model. Combining the perceptual–motor capacities of EPIC and the problem-solving capacities of SOAR (Newell, 1990), they provided a model that learns to solve conflicts in using resources. For example, if both tasks in a dual-task situation want to use the visual system, SOAR uses its problem-solving architecture to resolve this conflict, and learns new rules to avoid the conflict on future occasions.

Each of the three projects offers a partial solution to the learning problem. Logan's (1988) instance theory is consistent with the minimal control principle, because it reduces an algorithmic process with potentially many control states to an instance process with just one control state. However, it is restricted to situations in which a skill can be performed by a single retrieval from memory. Kieras et al.'s (2000) solution has the potential to decrease the number of control states, but lacks a learning mechanism. The EPIC–SOAR solution also cuts out some control states by learning rules that serve as shortcuts for longer chains of problem solving, but it is currently limited to solving resource conflicts.

2. A model of perfect time-sharing

This is our first example of a model that learns from instructions and uses the minimal control principle to achieve perfect time-sharing. We interleave the discussion of the model with an explanation of the relevant parts of the ACT–R (Anderson et al., 2004) architecture in which it is implemented.

Although ACT–R is in its roots a production system, it has evolved away from the classical paradigm. The latest developments are strong connections to functional MRI data (e.g., Anderson, 2005, this issue), reduction of the expressive power of production rules to make them more neurologically plausible, a more parsimonious mechanism for learning rules, and perceptual and motor modules from EPIC that interact with the outside world. Also, some traditional components of production systems have been removed, such as the use of a goal stack, because they produced predictions inconsistent with human data.

2.1. The central production system, modules, and buffers

ACT–R is designed to interact with software running the experiment or task. It can direct visual attention to a part of the screen and "see" what is there. Similarly, an ACT–R model can initiate a motor action such as pressing a certain key, which is then transmitted to the experimental program. This interaction with the world is carried out by specialized modules that make their results available to *buffers,* which are also used to issue new commands to the modules, such as the initiation of a key stroke or eye movement. Fig. 1 illustrates this concept. At the heart of this diagram is a production system. The rules collect their information not only from the perceptual–motor buffers, but also from other internal buffers and modules. Only the visual and the manual systems are depicted in the diagram, but ACT–R also implements an aural and vocal system. To retrieve a certain fact from declarative memory, a request must be made to declarative memory in the form of a partially completed pattern. The declarative module then tries to complete the pattern, after which the result is placed back in the retrieval

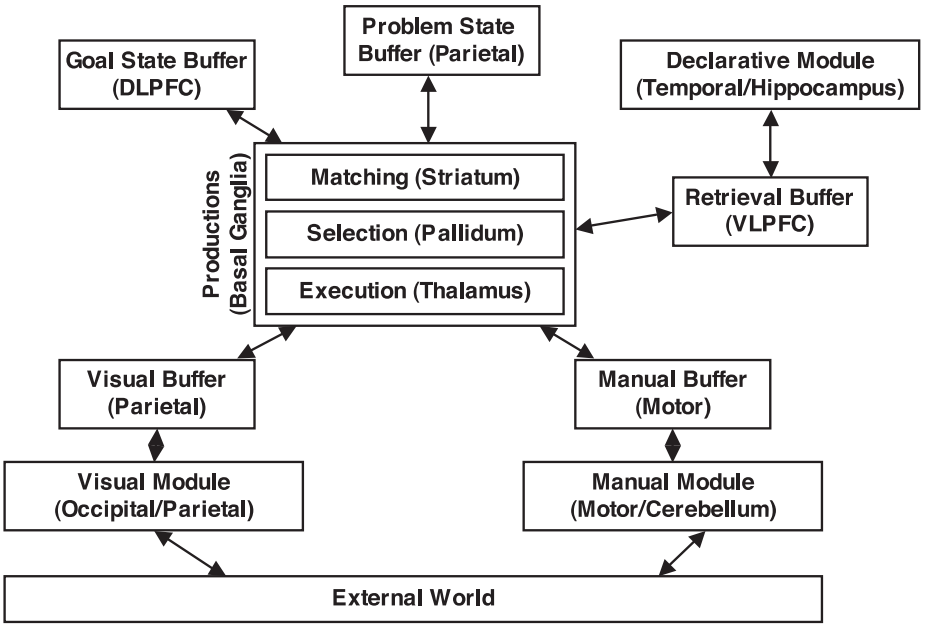

Fig. 1. Overview of the ACT–R architecture (adapted from Anderson et al., 2004).

buffer, where it can be matched and used by another rule. Another example is the visual system: To attend to a location, the production system must make a request to the visual system through the visual buffer. For our present discussion, the goal state buffer[1] is important, because it holds this control state. This means that the minimal control principle can be stated very precisely: *The model that is least susceptible to brittleness is the one with the fewest possible values in the goal state buffer.*

Production rules in ACT–R serve a switchboard function, connecting certain information patterns in the buffers to changes in buffer content, which in turn trigger operations in the corresponding modules. The advantage of this restricted power of rules is that modules can operate in parallel and relatively independent of each other. Behavior within a module is largely serial. For instance, the declarative model can only retrieve one item at a time, and the visual system can only focus its attention on one item in the visual field at a time. Timing within a module is usually variable and depends on the module. The central production system can also execute only one rule at a time, taking 50 msec per rule.

The perfect time-sharing experiments that are the topic of this section were reported by Schumacher et al. (2001). Participants performed a visual–manual task and an aural–vocal task concurrently. For the visual–manual task, a circle appeared in one of three horizontal locations to which they made a spatially compatible response with their right hand, pressing their index, middle, or ring finger to left, middle, or right locations. For the aural–vocal task, participants were presented with tones that lasted 40 msec and were either 220 Hz, 880 Hz, or 3520 Hz, to which they responded "one," "two," or "three." The experiment consisted of homogeneous single-task blocks, in which participants did just the visual–manual task or just the aural–vocal

task, and mixed-trial blocks. The mixed-trial blocks consisted of dual-task trials, in which both stimuli were presented simultaneously, and heterogeneous single-task trials, in which only one of the two stimuli were presented. The participant would therefore never know whether just one or both stimuli would be presented in a trial.

Byrne and Anderson (2001b) modeled this experiment at the level of expert behavior. Instead of presenting their model, we walk through the expert level of our own model, which is almost identical. Fig. 2 gives a time diagram of the model's execution. The left-hand side represents the moment that both stimuli are presented, and time progresses to the right. A block in the diagram represents processing in one of the modules and arrows indicate dependencies between them.

1. First, the visual module notices that a new stimulus has appeared on the display. It responds to this by updating the visual buffer, noting that a new stimulus is in the visual field, and giving its location. This ACT–R assumption is that this is instantaneous, so there is no box in the diagram.
2. A production rule (compiled-attention-rule) notices the update in the visual buffer and puts a request in the visual buffer to attend the new visual stimulus. At the same time the aural module detects the tone and updates the aural buffer to indicate this.
3. A production rule (attend-aural) puts in a request to identify the tone that has been detected by the aural module.
4. The visual module finishes the eye movement and encoding of the visual stimulus, having found the pattern "O – –," which it places in the visual buffer.

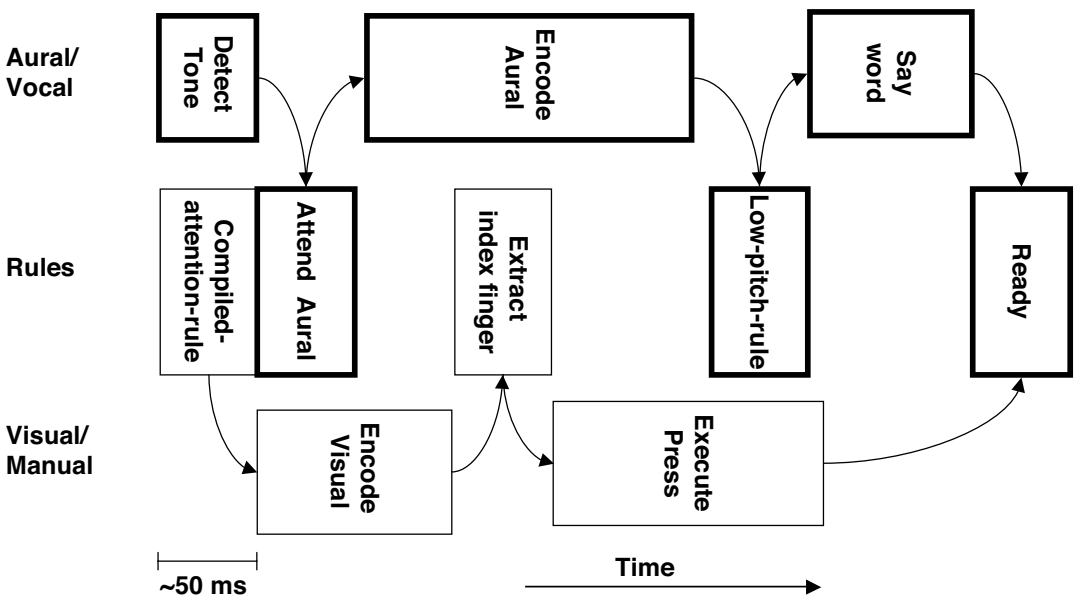

Fig. 2. Time diagram of expert-level parallel behavior in the Schumacher et al. (2001) Experiment 1 according to the ACT–R model. Arrows indicate dependencies between steps. Thin-bordered boxes represent steps in the visual–motor task and thick-bordered boxes in the aural–vocal task.

5. A production rule (extract-index-finger) notices the "O – –" pattern in the visual buffer and puts in a request to press the index finger in the manual buffer.
6. The motor module notices the request in the manual buffer and starts initiating the key press.
7. The aural module finally determines that the tone is a low pitch and places this information in the aural buffer.
8. A production rule (low-pitch-rule) notices that a low pitch has been detected and puts a request to say "One" into the vocal buffer.
9. The vocal module starts processing the request to say "One."
10. When both the vocal and the module are done, the rule "Ready" terminates the trial.

In the timing diagram, the thin-bordered blocks are steps in the visual–manual task, whereas the tick-bordered blocks are steps in the aural–vocal task. There are no gaps between blocks of each of the two tasks, indicating that the time-sharing is perfect.

2.2. The role of declarative memory

Up to this point we have discussed the contributions of the perceptual–motor systems and the central role of the rules. To explain how the highly specialized rules can be learned, two additional components are needed: declarative memory and a mechanism to learn new rules from its contents. Declarative memory holds information in a fairly literal form. It can be used to store all kinds of knowledge, including facts (such as 3 + 4 = 7), old experiences (the light switch in the closet is not operational), and instructions (boil water to make tea). Declarative knowledge is passive: It is stored and will only be brought to bear if it is requested by a production rule action. The process is similar to the interaction with the perceptual–motor systems: A production rule issues a request to declarative memory for a certain fact by placing part of the fact as a cue in the retrieval (declarative) buffer (i.e., 3 + 4 = ?, the light switch in the closet ?). The declarative memory module then tries to complete the pattern. This completion attempt fails if no matching memory is available or if the matching memory pattern has decayed too much. The exact characteristics of declarative memory in ACT–R can be found in Anderson et al. (2004). What is important for this discussion is that strong memories are recovered very quickly from declarative memory, whereas weak memories are recovered more slowly or not at all.

The assumption for the model of the Schumacher task (Schumacher et al., 2001) is that the instructions are stored in the following declarative form:

1. Attend any aural stimulus.
2. Attend any visual stimulus.
3. Given an attended aural stimulus, retrieve and make a response for that stimulus.
4. Given an attended visual stimulus, determine and make a response for that stimulus.

The assumption in these instructions is that to process a stimulus, either visual or aural, it first has to be attended, that is, the pitch of the tone or the identity of the visual stimulus must be determined. This is taken care of by Instructions 1 and 2, whereas Instructions 3 and 4 prescribe what should be done with a stimulus once it has been identified. For the aural stimulus, this re-

quires retrieving a mapping from declarative memory (i.e., low-pitch maps onto "one") and then giving that response. For the visual stimulus, there is a straightforward mapping between location and finger. We therefore assume that no memory retrieval is necessary to determine this mapping.

As declarative knowledge is passive by itself, the instructions given previously must be retrieved, interpreted, and carried out by production rules. This is much slower than the process with the expert model. This has consequences for the optimal order in which the instructions should be carried out, because as long as the model is in the instruction interpretation stage, dual-task costs are incurred regardless of the execution order. It is therefore almost impossible to initially determine the optimal order, because this might change later due to learning. Also, given that some of the trials in a mixed block are single-task trials, not all of the instructions necessarily have to be carried out on each trial, frustrating a plan that waits for a stimulus that never appears. Here we invoke the minimal control principle: Instead of having states for the different stages in the process, one state is sufficient, because the instructions themselves contain information on when they are applicable. The only thing that is needed are interpretation rules that retrieve the right instruction at the right time.

To retrieve the proper instruction at the right moment, there are four retrieval rules for each of the possible event types, such as

Get-next-visual-location-instruction-rule
IF the goal is to do a certain task (goal buffer)
AND the visual module has detected a new stimulus (visual buffer)
THEN request an instruction to do something with a visual stimulus in the context of the current task (retrieval buffer)

A second rule then executes an instruction, such as the one to attend the visual stimulus:

Attend-visual-stimulus
IF the goal is to do a certain task (goal buffer)
AND the instruction is to attend a visual stimulus (retrieval buffer)
AND there is a location in the visual buffer (visual buffer)
THEN put a request in the visual buffer to attend that location (visual buffer)

In total, the model has seven execution rules, one to attend to a visual stimulus, one to attend to an aural stimulus, one rule to retrieve a pitch-word combination, one rule to say the word that has been retrieved, and three rules that implement the mappings of the visual stimuli on the three fingers that are used to make the response. The assumption is that because the stimulus-response mapping between the visual stimuli and the response is compatible, no retrieval from declarative memory is necessary. The following encoding parameters were estimated: 50 msec to recognize that there is a tone (which is the ACT–R default), 200 msec to determine the pitch of the tone (default is 285 msec), and 100 msec to attend the visual stimulus (default is 85 msec). For verbal and motor responses, ACT–R's default values were used. Fig. 3 shows a diagram of the behavior of the model as a novice. Each step consists of at least three substeps: a rule that retrieves the next instruction, the retrieval of the instruction, and the execution of the retrieved instruction. The steps that are part of the visual–manual task are in the thin-bordered boxes. We can derive that some dual-task costs occur from

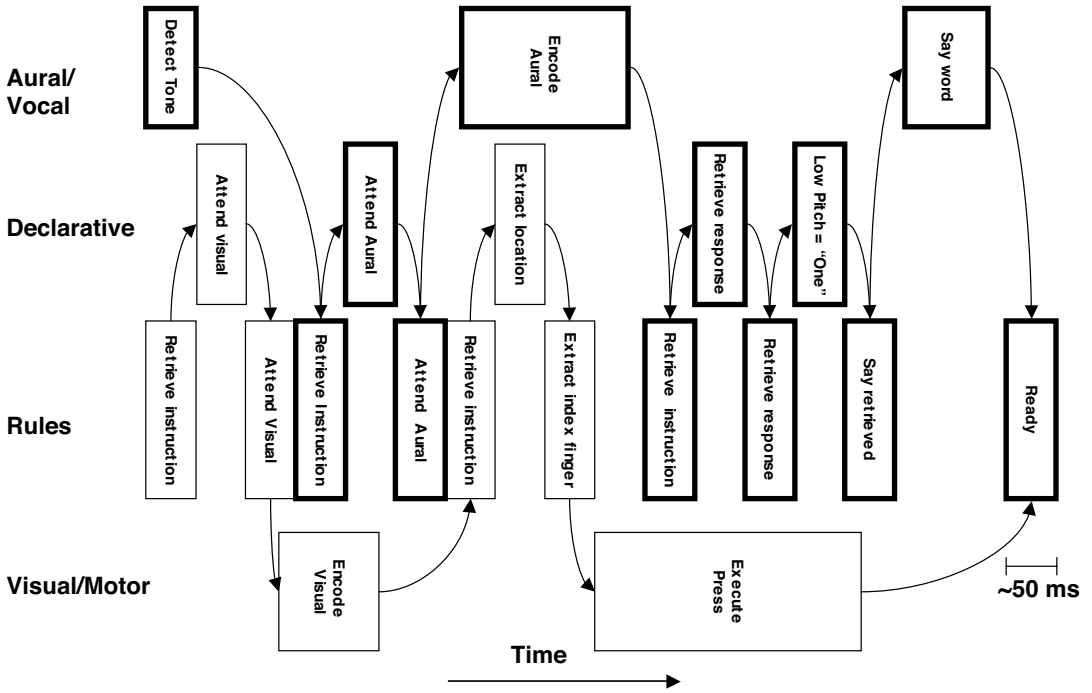

Fig. 3. Time diagram of a novice in the Schumacher et al. (2001) Experiment 1 according to the ACT–R model.

the fact that not all thin-bordered boxes are horizontally next to each other: the time between the encoding of the visual stimulus and the retrieval of the instruction that extracts the position of the circle from that stimulus. The aural–vocal task (represented by thick-bordered boxes) also has a dual-task cost because it must wait for completion of the initial steps that handle the visual stimulus.

2.3. From declarative to procedural knowledge: Production compilation

To explain how novice behavior can develop into expert behavior through training, one final component of ACT–R must be described: *production compilation,* the mechanism for learning new production rules. Production compilation (Taatgen & Anderson, 2002) learns new rules by combining two existing rules that fire in sequence into one new rule. If the first of the two rules makes a request to declarative memory, the result of which is used by the second rule, then the retrieved chunk is substituted into the new rule, effectively eliminating the retrieval. In itself this mechanism only produces more efficient and specialized representations of knowledge that are already available, which is sufficient for our present purpose. However, when the production rules that are compiled are a general cognitive strategy, the resulting rules, although specializations of the general strategy, nevertheless generalize the specific experience.

An example of production rule learning is the combination of the rule *Get-next-visual-location-instruction-rule* and *Attend-visual-stimulus* shown earlier. When they are compiled with the instruction to attend any visual stimulus, ACT–R learns the rule:

Compiled-attention-rule
IF the goal is to do the Schumacher task (goal buffer)
AND the visual module has detected a new stimulus (visual buffer)
THEN attend that stimulus (visual buffer)

Newly learned rules can be the source of recombination themselves, enabling the collapse of more than two rules into one. For example, the following rule summarizes the processing from three rules and two retrievals, and it was learned by first compiling the first two rules with the retrieval of the instruction and then compiling the resulting rule with the final rule and the retrieval of the response set (low-pitch—"one"):

Learned-Low-Pitch-rule
IF the goal is to do the Schumacher task (goal buffer)
AND a tone with a low pitch has been detected (aural buffer)
THEN say "one" (vocal buffer)

This rule, like the earlier compiled rule, is state independent: It reacts to the fact that a tone with a low pitch has been detected. As a result, the rules that make up expert behavior produce reactive, bottom-up behavior: Steps are not planned but cued by the environment.

After all the steps in the novice model have been combined, the behavior depicted in Fig. 2 is reached. The expert behavior shown in Fig. 2 involves no more dual-task costs, because there is no interference between the two tasks. Of course, not all combinations of rules can be compiled. If a first rule makes a request to a perceptual module and the second rule acts on the result from the module, the rules cannot be compiled because the outcome depends on external input. Similarly, if two rules send a request to the same module (e.g., two consecutive key presses), they also cannot be compiled. As a result, learning eventually produces rules that move the system from one perceptual input or motor output to the next (as far as they depend on each other), making perceptual and motor actions the main determiner of performance. This model indeed reaches this stage (Fig. 2), but if a task is more complex, there may never be enough time to learn all the possible rules that are needed.

2.4. Gradual learning of production rules

If we were to introduce these new rules right away into the system, expert behavior would be reached in a few trials, which obviously does not occur with humans in the Schumacher task (Schumacher et al., 2001). Anderson, Fincham, and Douglass (1997) also found evidence that procedural learning is a slow process. To account for this finding, instead of introducing these new rules into the production system right away, ACT–R gradually phases them in by using *production utility,* the subsymbolic component of production rules. Utility is calculated from estimates of the cost and probability of reaching the goal if that production rule is chosen, with the unit of cost being time. ACT–R's learning mechanisms constantly update the parameters

used to estimate utility based on experience. If multiple production rules are applicable for a certain goal, the production rule with the highest utility is selected. This selection process is noisy, so the rule with the highest utility has the greatest probability of being selected, but other rules have an opportunity as well. This strategy may produce errors or suboptimal behavior, but it also lets the system explore knowledge and strategies that are still evolving.

When a new rule is learned, it is given a very low initial estimate of utility. This value is so low that the new rule effectively has no chance of being chosen, because it must compete with the parent rules from which it was learned. To have a reasonable chance of selection, the rule must be recreated a number of times. This has the advantage that only new rules which correspond to frequent reasoning patterns will make it into the effective rule set. More formally, utility has two components, one that it inherits from the parents and another based on the experiences that the rule itself gains, combined in the formula

$$U_p = \frac{mU_{p,\text{prior}} + \text{experiences} \cdot U_{\text{experienced}}}{m + \text{experiences}}$$

In this equation, $U_{p,prior}$ represents the utility component inherited from the parent rule, which starts out very low but which is increased at each recreation of the rule, and $U_{experienced}$, the average of the experienced utility of the rule (which will only have a nonzero value if there have been experiences). Each component is scaled, $U_{p,prior}$ by m, a fixed parameter that is set to 10 by default, and $U_{experienced}$ by the number of actual experiences. As a consequence, the impact of $U_{p,prior}$ is large initially, but decreases in importance as the rule gains its own experiences. $U_{p,prior}$ is set to –20 when the rule is first created, but after each recreation its value is increased toward the value of its parent's utility according to the equation

$$U_{p,prior}(t+1) = U_{p,prior}(t) + \alpha[U_{parent}(t) - U_{p,prior}(t)]$$

The new value of $U_{p,prior}$ is made equal to its old value plus a fraction of the difference between its current value and the utility of its parent. As a result, the utility value of a new rule will initially increase toward its parent's value quite steeply, but then levels off as it approaches the parent's value. However, as it approaches the parent's value, the new rule gets a reasonable chance of winning the competition and gaining experiences itself, because selection is a noisy process.

Fig. 4 gives an example of how this works in practice based on the parameters used in the model, depicting the growth of utility for a new rule and the competition with its parent. In this example, the utility of the parent rule is 10, and the actual utility of the new rule is 10.1, indicating a 100 msec gain (it saves one production rule and one retrieval), but the model can only discover this by having experience with the rule. The x axis represents opportunities for the new rule to fire, with the creation of the rule at 0. The rule starts with a utility of –20, and thus has zero probability of winning the competition with its parent. When the parent wins, however, the new rule is recreated, increasing its utility. At around 100 recreations, the new rule starts having a small probability of winning the competition and therefore gaining its own experiences. At some later point, the new rule's utility surpasses that of its parent due to experiences with a utility of 10.1, eventually making it the dominant rule. In terms of reaction times, this particular rule produces a speedup of 100 msec that starts building up at around 100 experi-

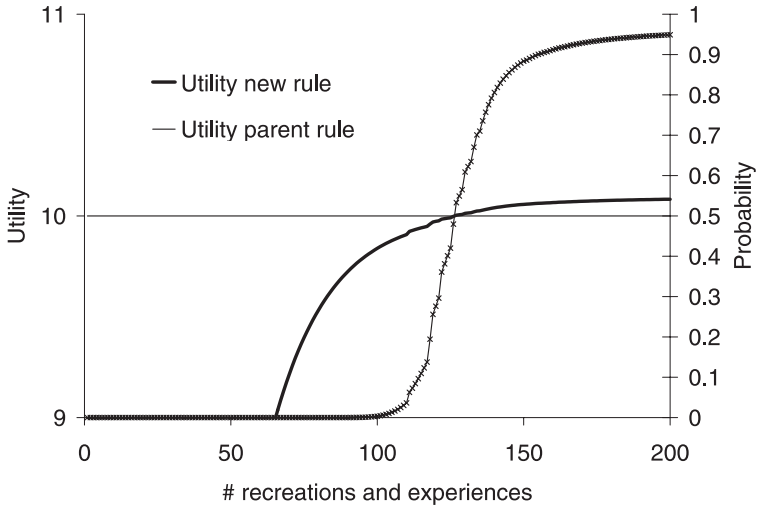

Fig. 4. Example of gradual introduction of a new production rule. The thin line represents the utility of the parent rule, and is fixed on 10. The new rule starts with a utility of –20 represented by the thick line (which is initially outside the graph's scale), but eventually approaches its true value of 10.1. The S-shaped curve represents the probability (with its axes on the right side of the diagram) that the new rule will win from the parent rule.

ences and reaches a peak around 150 experiences, where it wins from its parent around 95% of the time. This mechanism of gradual introduction of new rules produces a slow transition from the behavior in Fig. 3 to the behavior in Fig. 2. A consequence of the gradual introduction mechanism is that there is always a small residual probability that the parent is selected instead of the new rule, occasionally producing a suboptimal trial. We now discuss the three Schumacher et al. (2001) experiments in detail and compare the model's outcome to the data from the participants.

2.5. Experiment 1

In Schumacher et al.'s (2001) first experiment, pure-trial blocks consisted of 45 trials each, whereas mixed-trial blocks consisted of 48 trials, 18 dual task and 30 single task. On Day 1 of the experiment, participants were trained just on pure-trial blocks, 6 of each type. Day 2 consisted of 6 pure-trial blocks and 8 mixed-trial blocks, and Days 3 to 5 consisted of 6 pure-trial blocks and 10 mixed-trial blocks. Fig. 5 shows the reaction times for each of the subtasks of the experiment for both participants and the model. On Day 2 there is a clear dual-task cost for the aural–vocal task, because it is slower than both single-task variants. The dual-task cost for the visual–manual task is almost nonexistent, suggesting that, on Day 2, participants complete the visual–manual task before the aural–vocal task. On Day 5, however, all dual-task costs have disappeared. The model nicely replicates all these effects and also reaches a stage in which no dual-task costs are incurred. The residual probability that a parent rule is selected instead of the optimal rule is small in this model (<1%), and even then it does not affect dual-task costs because there is still some "slack" in Fig. 2 to compensate for it.

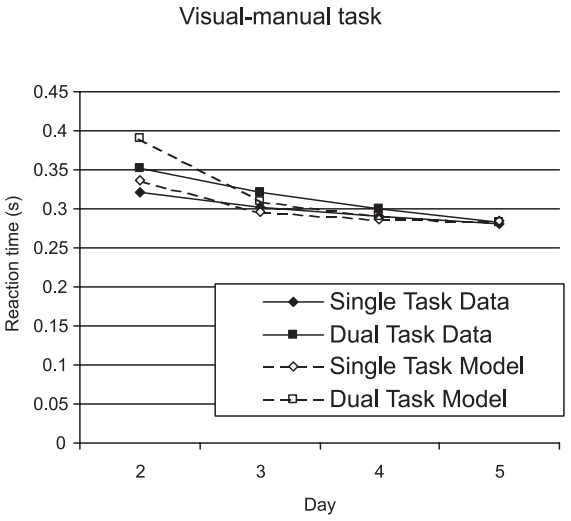

Fig. 5a. Results of Experiment 1 of Schumacher et al. (2001), model and data, visual-manual task. Single-task trials are averages of homogeneous and heterogeneous trials.

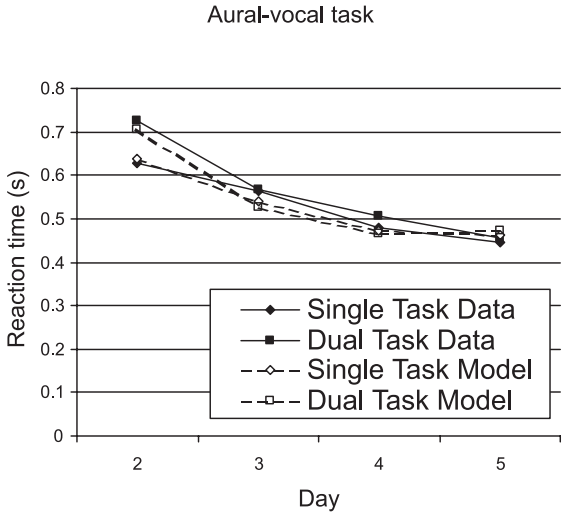

Fig. 5b. Results of Experiment 1 of Schumacher et al. (2001), model and data, aural-vocal task. Single-task trials are averages of homogeneous and heterogeneous trials.

2.6. *Experiment 2*

According to Schumacher et al. (2001), dual-task costs in psychological refractory period experiments are often due to ordering instructions. To further show this, they took the participants from Experiment 1 and gave them additional blocks of trials, but now with the explicit instruction to always give the vocal response before the motor response. The visual stimulus was presented

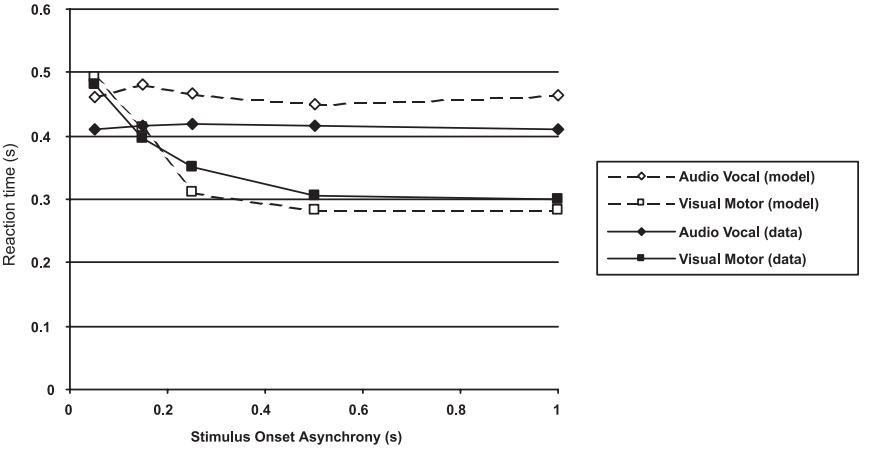

Fig. 6. Results from Experiment 2 of Schumacher et al. (2001), model and data.

50, 150, 250, 500, or 1,000 msec after the aural stimulus, the stimulus onset asynchrony (SOA) that is typical for psychological refractory period experiments. The model achieved this by introducing a single top-down constraint to the rules that issue the motor commands, by specifying that the manual response could only be initiated after the initiation of the vocal response. Fig. 6 shows the results of the experiment and of the model simulation. The primary task, the aural–vocal task, shows no effect of SOA and has no additional costs due to the second visual–motor task. The visual–motor task shows dual-task costs for SOAs of 250 msec and smaller, because the manual response must wait until the vocal response has been initiated.

2.7. Experiment 3

Experiments 1 and 2 have made plausible that dual-task costs can at least sometimes be attributed to instructions. However, Schumacher et al. (2001) were looking for evidence consistent with the absence of a central bottleneck, that is, for the hypothesis that processing in central cognition occurs in parallel. Because the visual–manual task is so much easier than the aural–vocal task, serial central processing could not be ruled out. Indeed, the ACT–R model has a central cognitive bottleneck and is perfectly capable of modeling the data. To make the visual–motor task harder, they added one extra stimulus and made the mapping incompatible. Now there were four possible positions for the visual stimulus, and the mapping of the positions on the finger was reversed—for example, for the right-most position the left-most finger had to be pressed. The experiment was otherwise identical to Experiment 1, except that it was extended to 6 days. We modified the model to accommodate the new task. First, determining which finger corresponds to what location can no longer be considered automatic, so we changed the instruction to determine the finger to be pressed to retrieve a mapping between location and finger from memory. Second, because the response selection moments are much closer in time than in Experiment 1, we added noise to the perceptual encoding times to make more accurate predictions, so the time to determine the pitch of the tone varied from 0.1 to 0.3 sec, and the visual encoding time varied from 0.1 to 0.2 sec. Finally, we increased the duration

of the production rule that initiates the motor output from 0.05 to 0.2 sec. The reason for this last change was that incompatible mappings have a lasting effect on the response time that the model cannot explain. Although we have some ideas on how to accommodate this matter, it is not the topic of this article, so we modeled the difference parametrically. The results (Fig. 7) show that, in both the model and the experiment, there are some residual dual-task costs, although the model only shows them in the aural–vocal task. A further analysis of Schumacher et al. showed that 5 of the 11 participants had virtually no dual-task costs by the end of the experi-

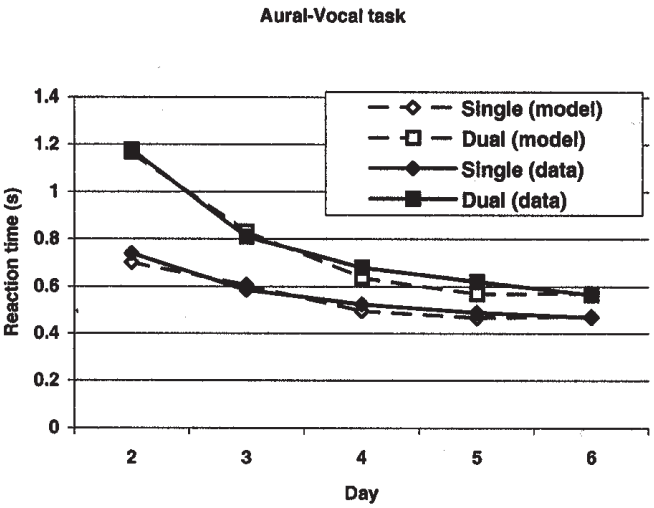

Fig. 7a. Results of Experiment 3, model and data, aural-vocal task. Single-task trials are averages of homogeneous and heterogeneous trials.

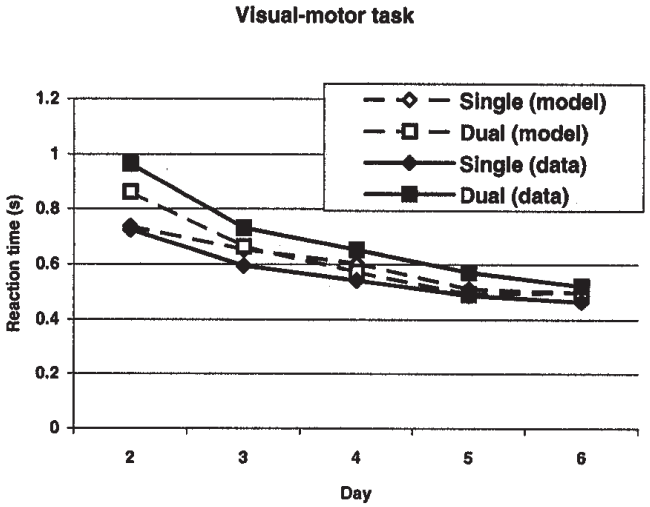

Fig. 7b. Results of Experiment 3, model and data, visual-motor task. Single-task trials are averages of homogeneous and heterogeneous trials.

ment. These participants were also the fastest on the task. When the duration of the production rule that initiates the motor response is lowered from 0.2 sec to 0.15 sec, the model reproduces this effect. Also, when this value is increased to 0.35 sec, the model produces the behavior of the 4 participants who had large dual-task costs (Fig. 8). This suggests that the ability to handle the incompatibility in the mapping between the visual stimulus and the finger to be pressed can explain the observed individual differences. However, more solid evidence is needed to corroborate this finding.

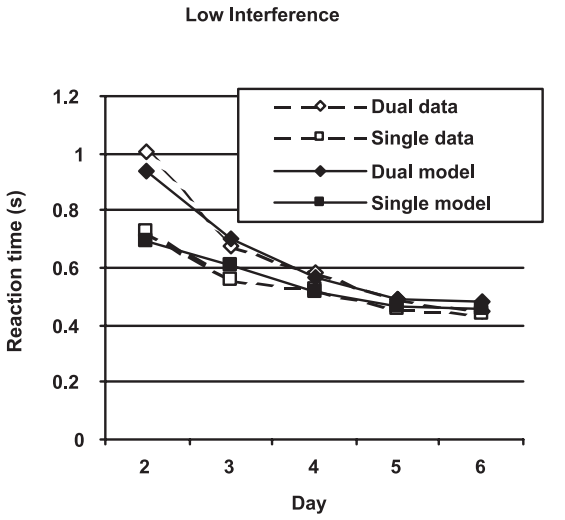

Fig. 8a. Data and model for 5 participants with no dual-task costs (low interference). Results are averaged over the two tasks and the two types of single-task trials.

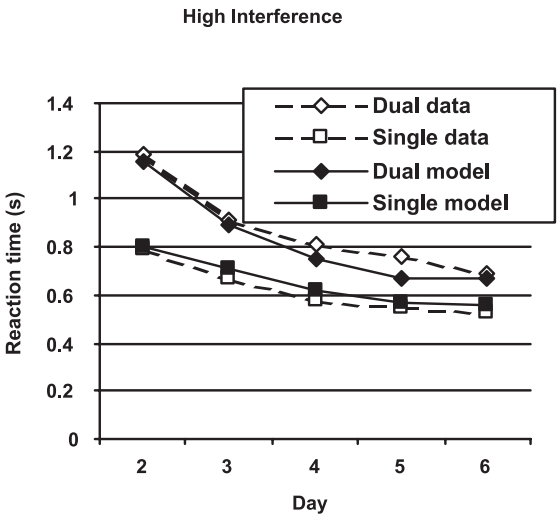

Fig. 8b. Data and model for 4 participants with high dual-task costs (high interference). Results are averaged over the two tasks and the two types of single-task trials.

2.8. Discussion

The models of the Schumacher et al. (2001) experiments start out with a set of instructions that must be retrieved when needed. Eventually they learn production rules that are sensitive to the conditions under which they should be applied, which are usually independent of order, with the exception of Experiment 2 in which the motor response must be made before the vocal response. This order independency is an important aspect of the ability to parallelize behavior. ACT–R's central cognitive processes are serial, but firing a single production rule only takes 50 msec. If all relevant knowledge is fully proceduralized and ready to respond to the conditions in which it is applicable, this restriction will hardly ever be the bottleneck of processing. When knowledge is still declarative, the duration of the bottleneck increases, because retrieving and carrying out an instruction takes a significant amount of time.

It would have been very hard to model this task with a more hierarchical representation of instructions with top-down control, unless an explicit general executive is added that schedules all the steps. The proper order of carrying out the instructions would then be a matter of planning, which can only be done when the instructions are proceduralized, because only then can it be determined how much time they take. This model is driven purely by the perceptual inputs and ends up with a greedy strategy that chooses any step that it can carry out at the moment. Although a greedy planning strategy is not optimal in general, it is for this task. We can therefore conclude that the minimal control principle is useful for a small task, both in describing human behavior and in prescribing how a model of that behavior should be constructed. Whether a greedy strategy is always the best solution, both in the descriptive and prescriptive sense, remains to be seen.

Hazeltine, Teague, and Ivry (2002) conducted a further series of dual-task experiments in an effort to rule out a central bottleneck. With models similar to the ones discussed here, Anderson, Taatgen, and Byrne (in press) demonstrated that a central bottleneck model can still model these more intricate experiments and can even explain subtle dual-task effects in some of them.

3. Skill acquisition in complex dynamic tasks

Principles derived from simple tasks should scale up to real-world cognition, and we therefore shift our attention to a real-time, complex dynamic task. Although it does not involve any explicit multitasking, there are many opportunities for what could be called *within-task multitasking*. In most of the examples we examine, this within-task multitasking consists of two instructions that are initially carried out in sequence, but that start overlapping with experience.

The CMU-ASP task is a simplification of Georgia Tech Aegis Simulation Program (Hodge et al., 1995), in which the participant takes the role of an operator on a ship who must use a radar screen that displays various tracks of airplanes. The goal is to classify these planes, which involves asking the system for information and entering the classification by a sequence of keystrokes. Fig. 9 shows the interface to the system. The circular area that makes up most of the screen is the radar scope. The small circle in the middle represents the home ship, and the rect-

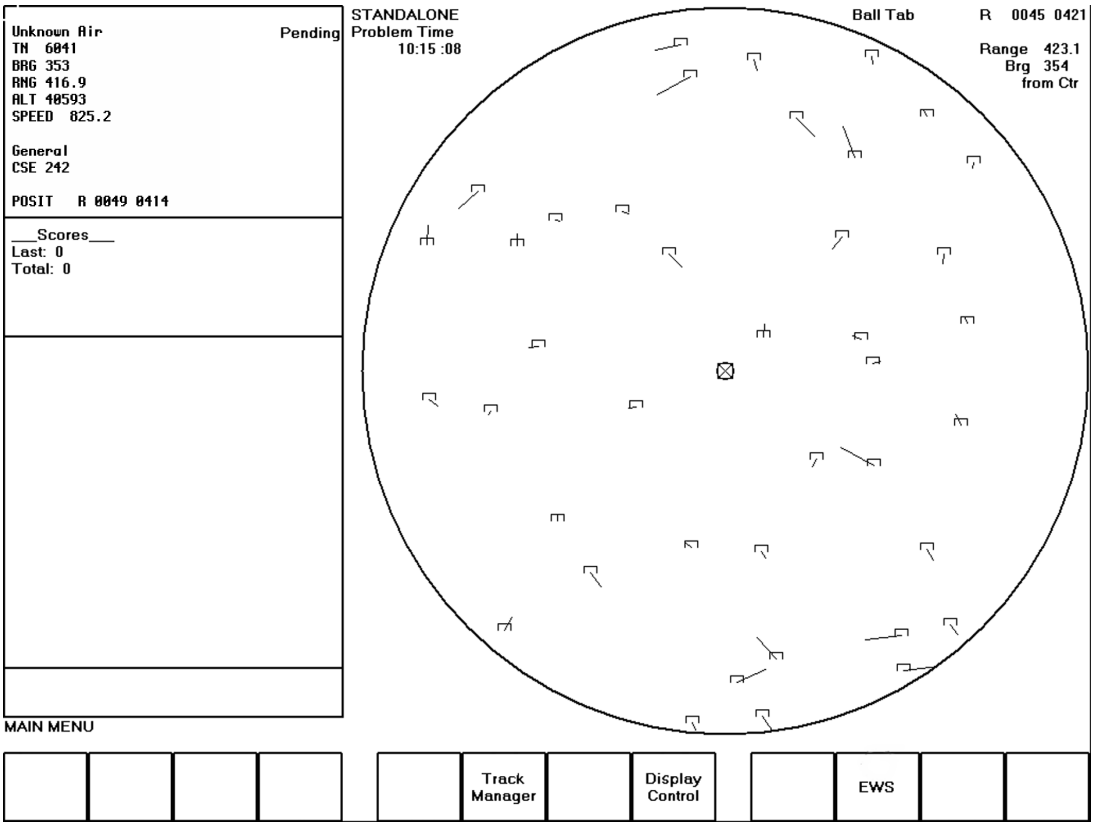

Fig. 9. The CMU-ASP task.

angles with lines emanating from them represent unidentified planes. Participants can select a plane to work on by clicking it with the mouse, after which some information about that plane will appear in the top-left area of the screen. The information important to this task is the altitude (ALT) and speed (SPEED), which are sometimes enough to determine the identity of a plane. A third region of importance is the bottom row of 12 boxes, which correspond to the 12 function keys on the keyboard. For example, to activate the Track Manager, a menu choice to enter airplane classifications into the system, the F6 key must be pressed because the label "Track Manager" is in the sixth box from the left. Labels change dynamically depending on the context: Once F6 has been pressed, a new set of functions is displayed in the bottom row. Because the labels in the bottom row change, mastery of this aspect of the task requires considerable learning.

To do the task, participants must take three steps for each plane. The first involves selecting a plane to work on, which is accomplished by clicking on one of the unidentified planes on the radar screen. Although any unidentified plane will do, the instructions specify some preference relations: Planes that are close to the home ship (the middle of the scope), planes that are fast (as identified by length of the line emanating from the rectangles), and planes that are headed for the home ship (when the line from the rectangle is pointing toward the center circle) should

be classified first. Step 2 concerns determining the identity of the plane, for which there are two methods both of which are only sometimes successful. The first method is to look at the speed and altitude in the top left of the screen. If speed and altitude are within certain margins, the plane is a commercial airplane, otherwise the identity cannot be determined by that method. The second method is to use radar profiling, which is accomplished by giving the "EWS" command (by pushing F10) and confirming this command with a second key press. Radar profiling may give an answer that allows a direct classification, or it may return "negative." If neither classification method is successful, the participant should move to the next plane, otherwise they must proceed with Step 3: Enter the identity into the system via the Track Manager, which requires a sequence of eight key presses. For example, to enter a commercial airplane, one enters function keys corresponding to the labels: "Track Manager"—"Update hooked track"—"Class/Amp"—"Primary ID"—"Friend"—"Air Id Amp"—"Nonmilitary"—"Save Changes". For each successful and correct classification, points are awarded that are displayed in the middle left of the screen. The number of points depends on the priority of the plane and the time within the scenario that it is classified: A fast plane that is close by and headed toward the home ship will yield many points if it is classified early, but when it is classified later its score will be smaller. The scoring scheme encourages participants to identify the important planes early in the scenario.

Anderson et al. (2004) reported an experiment that consisted of 20 scenarios of 6 min each, divided over 2 days. Participants showed considerable improvement in both speed and the number of classified planes. The results can be found in Fig. 11, which we discuss along with the model's ability to match participants' behaviors.

3.1. From a strong to a weak hierarchy

Anderson et al. (2004) modeled learning on the CMU-ASP task using a declarative representation of instructions in combination with production compilation. The model uses a hierarchical decomposition of the task into instruction sequences that can be interpreted by production rules. Each sequence consists of a number of steps that are carried out in order, some directly and others by referring to another rule for which a subgoal is created. The model learns by compiling the retrieval of instructions into task-specific rules and by learning which labels correspond to which keys.

Although the Anderson et al. (2004) model is successful in modeling the performance improvement in terms of speedup at the level of the three subtasks, and it can also predict how long attention is focused on certain regions of the screen during these subtasks, it is limited by the representation of instructions. It can model the speedup in performance but not the additional flexibility and parallelism that are associated with skilled behavior. In contrast, we present an updated model that accounts for three examples of additional flexibility:

1. Selection of a plane. Although the instructions state which planes should be classified first, participants initially do not seem to make much of an effort to optimize their selection. As they gain more experience, they gradually pay more attention to this, resulting in an improvement in the quality of the selected planes. The model explains this shift by dual-tasking the instructions to find a plane and clicking on a plane.

2. Selection of the appropriate function key. With practice, participants no longer have to look at the function key labels at the bottom of the screen, but rather know which to press. The Anderson et al. (2004) model explains this by recalling past goals, but we provide an alternative explanation in which the dual-tasking of finding a label and finding the corresponding function key makes label finding obsolete.

3. Optimization of hand movements. Interaction with the task requires using both the keyboard and the mouse, which means that the right hand must be moved between the two. However, moving the hand at the moment that it is needed is inefficient. Instead, its use can be anticipated, so that, for example, the hand is on the mouse at the moment that a new plane must be selected. Another option is to use the left hand for the keyboard and the right hand for the mouse.

3.2. Minimizing control

In Anderson et al.'s (2004) CMU-ASP model, the cue for retrieving the next instruction is the instruction that has just been carried out. This creates a strict top-down control of behavior, because the model is always driven by internal representations. Every step in the task execution therefore corresponds to a control state, effectively maximizing the number. Our previous dual-task model breaks the sequencing of instructions by having only a single control state and by using perceptual events as cues, producing pure bottom-up control. However, the complexity of the CMU-ASP task does not allow for pure bottom-up control. To be able to model flexibility in the CMU-ASP task, a compromise between bottom-up control and top-down control is required, because the model not only must react to perceptual and motor events, but also keep track of where it is in the process. Our solution is to replace the strong hierarchical instruction structure with a weak one with as few control states as possible, again following the minimal control principle. More specifically, it consists of three components:

1. Instead of a hierarchical structure, instructions are organized in sets that correspond roughly to unit tasks. Each instruction set has one or more conditions that specify when it is applicable. Within each set, instructions are sequenced by pairwise association strengths. Association strength is a feature of ACT–R's declarative memory not yet explained that increases the activation of a declarative memory element associated with this goal. We assume that associations between subsequent instructions are learned while reading and memorizing the instructions. Once the model starts carrying out the instructions, the next instruction in the sequence will receive extra activation because its predecessor has just been executed.

2. The instruction retrieval mechanism tries to retrieve an instruction within this set, relying on ACT–R's activation mechanisms to access the right instruction, usually the next one. Other sources of activation can influence the selection as well, such as a perceptual event that activates an instruction to handle that event type.

3. The rules that carry out a retrieved instruction check whether any constraints on the instruction are met, because the retrieval process does not guarantee this. For each instruction, it may take several steps to carry them out depending on the situation. For example, an instruction to click on a plane will initiate a movement of the hand to the mouse if the hand is not already there. If the hand is on the mouse and the mouse pointer is on the plane, then it will immediately initiate a click.

Within this weak hierarchical representation, each instruction can be considered as a control state, because it more or less determines the next step. However, once instruction retrieval has been compiled into task-specific rules, the instructions are no longer retrieved; the control states dissolve and leave only a single control state for the entire rule set. As a consequence, once the model has compiled all the instructions into rules, it has as many control states as rule sets, giving a large improvement over the strong hierarchical representation, which has as many control states as individual instructions.

3.3. The CMU-ASP model and global results

The model of the task uses declarative instructions that are organized into the five rule sets outlined in Fig. 10. The first four sets correspond to the main unit tasks: Select a plane, use one of two strategies to find the classification, and then enter the classification into the system. Each of the sets has one or more preconditions (in the hexagons) that must be satisfied before the set itself is activated. Only one set is active at a time, which is represented in the goal state buffer, but in principle it should be possible to switch between sets if the task demands this (this is not the case in this task, but it is in Salvucci's [2005, this issue] driving task). The fifth set, select key, finds and presses a function key on the basis of a label. It is activated whenever a select key instruction is en-

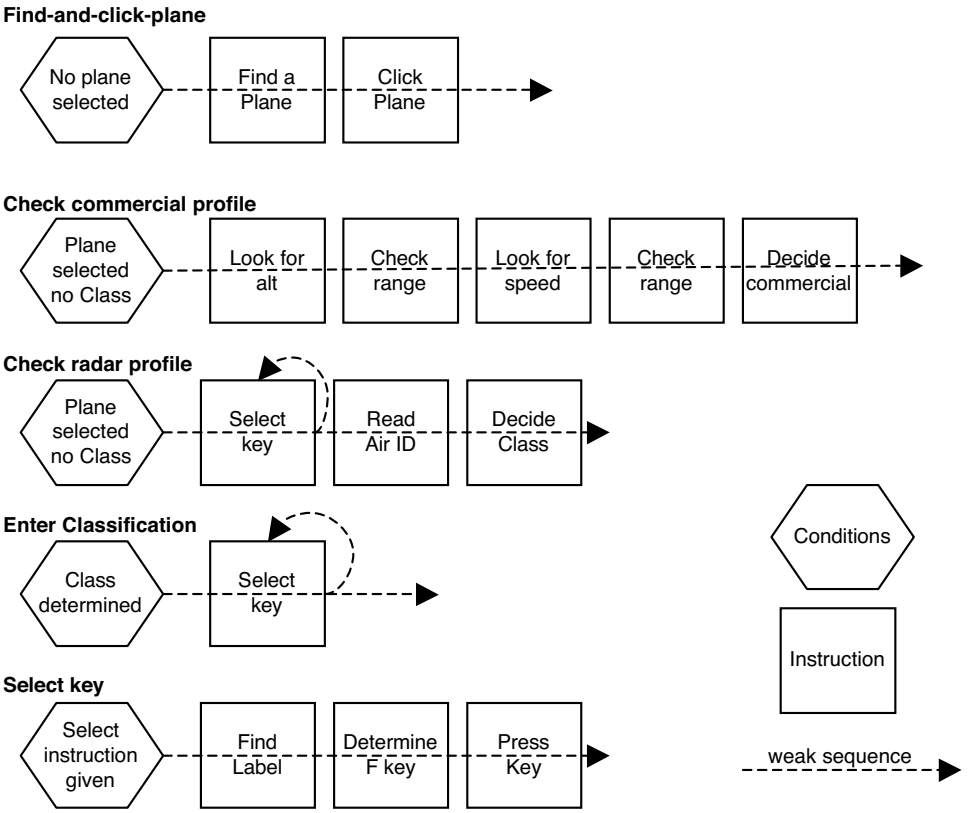

Fig. 10. Schematic representation of the instructions for CMU-ASP. Each row of boxes represents a rule set.

countered in one of the other sets, temporarily suspending that set. The instructions within a rule set are loosely ordered by activation. Instructions are retrieved by a rule and carried out by one or more subsequent rules. These subsequent rules also check the constraints on instructions. Production compilation combines instruction retrieval and interpretation rules into task-specific production rules that can fire out of sequence provided that their conditions are satisfied.

In the experiment reported by Anderson et al. (2004), participants had to solve twenty 6-min CMU-ASP scenarios spread over two sessions of 10 scenarios each. Fig. 11 shows the global behavioral outcomes of the experiment and the fit with the model. Fig. 11a shows the number of planes successfully classified in each scenario, whereas Fig. 11b shows the average time per scenario to perform one of three subtasks. *Select* is the elapsed time until a plane is clicked and corresponds to the *find-and-click-plane* rule set. *Classify* is the time between the mouse-click and the first keystroke of entering a classification (the "Track manager" keystroke) and covers two subtasks that cannot be separated easily: *Check-commercial-profile* and *Check-radar-profile*. Finally, *Enter* covers the time between the first and the last classification keystroke, that is, the sequence of eight keystrokes needed to enter the classification into the system. Although the general match between model and data is good, it is by no means better than that for Anderson et al.'s (2004) model. To see the difference, we have to examine the details of the new model's learning.

3.4. Finding and clicking a plane

The first step of identifying a plane is selecting a plane on the radar screen and clicking it. In the model these two instructions make up the first instruction set, which is applicable whenever there is no plane selected. The first instruction is to find a visual object of type *square-plane* on the screen and memorize it (*find-object*), and the second instruction is to click on the memorized object (*click-object*). Neither of these two instructions can be carried out in one step, in that both involve three steps. To do a *find-object*, first the location of an object of the appropriate type must be found on the screen. The next step is to initiate an eye movement to the object and to attend it. The third step is to process the information of the object and see whether it is a candidate for selection (when the track has been processed before, it is not a candidate). The *click-object* instruction requires first moving the hand to the mouse (if it is not already there), then moving the mouse onto the object, and finally clicking the mouse. Doing all these six steps (three from *find-object* and three from *click-object*) in sequence is inefficient, because the first three are perceptual actions and the last three are manual actions. Treating the execution of the two instructions as a dual-task situation (with restrictions) can lead to more efficient behavior. Initially the model will carry out all the steps in sequence by a retrieve-instruction and interpret-instruction cycle:

Retrieve-instruction
IF the goal is to do a rule set (goal buffer)
THEN retrieve an instruction from that rule set (retrieval buffer)

Implement-attend-location
IF the goal is to do a rule set (goal buffer)
AND the instruction is to attend a visual stimulus of a certain type (retrieval buffer)
AND the visual module has found an object of that type (visual buffer)
THEN move the eyes to that object and attend it (visual buffer)

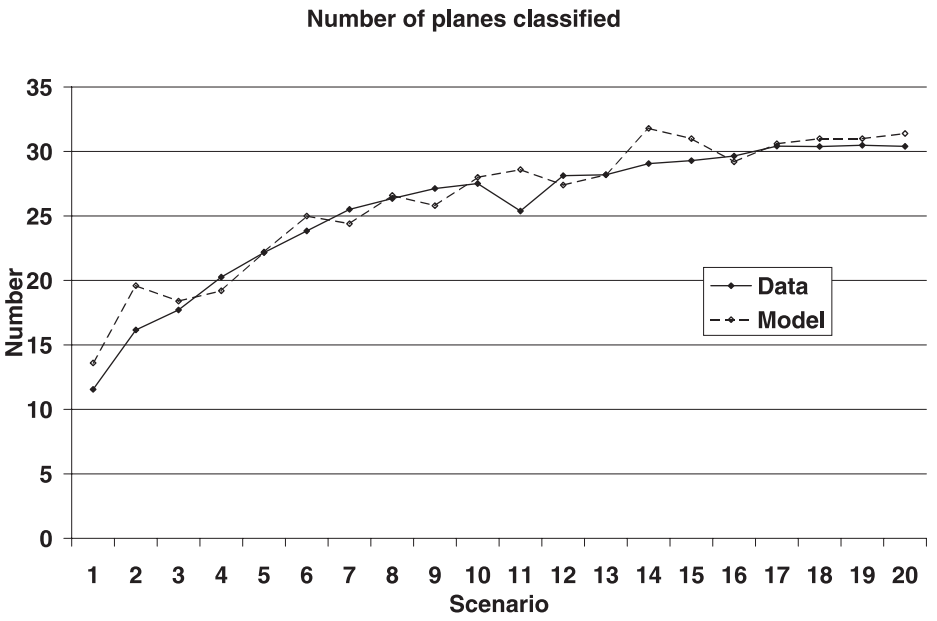

Fig. 11a. Results of the Anderson et al. (2004) experiment and the model: number of planes successfully classified per scenario.

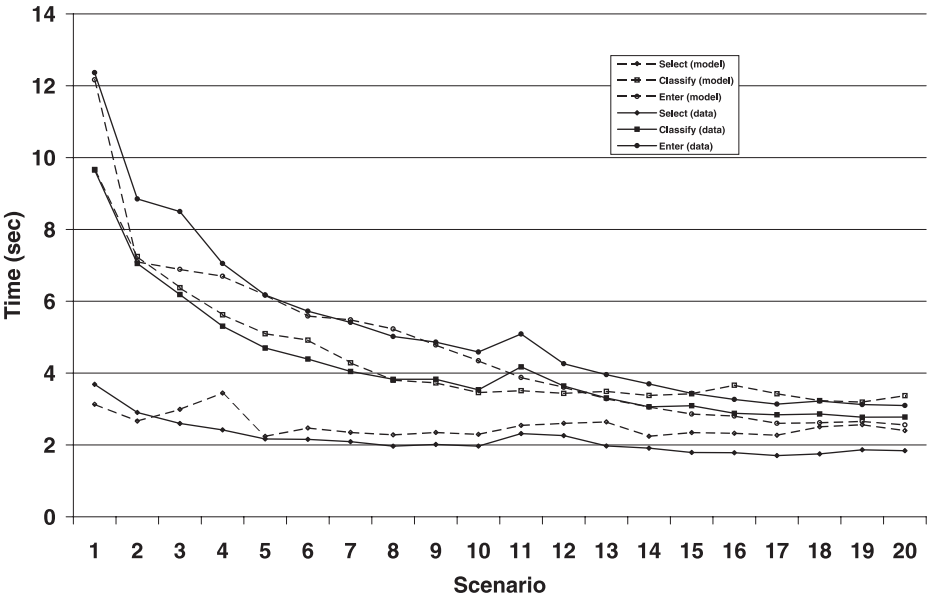

Fig. 11b. Results of the Anderson et al. (2004) experiment and the model: average time to complete a unit task per scenario.

The first of these rules will retrieve any instruction from this set. Activation between instructions will initially give *find-object* the highest activation, but, once *find-object* has been completed, *click-object* will have the highest activation. Although this activation scheme initially produces the same behavior as pure top-down models, the behavior changes once production rules are learned. The previously mentioned second rule is an example of a possible implementation rule. On the basis of the previous two rules in combination with the instruction to attend a square object, learning produces the rule:

Learned-attend-rule
IF the goal is to do rule set find-and-click-plane (goal buffer)
AND we have found a location on the screen of type square-object (visual buffer)
THEN move the eyes to the location and attend it (visual buffer)

Similarly, the new model learns a rule to move the hand to the mouse:

Learned-hand-to-mouse-rule
IF the goal is to do rule set find-and-click-plane (goal buffer)
AND the hand is not on the mouse (manual buffer)
THEN move the hand to the mouse (manual buffer)

A property of both rules is that they can activate at any moment when the *find-and-hook-plane* rule set is active and a location of the right type is found, or when the hand is not on the mouse. Although performance was initially constrained by the sequenced retrieval of instructions, they now depend only on the constraints in the rule and this control state. This lets the model treat the two instructions as dual tasks. For example, the *learned-hand-to-mouse-rule* can fire as soon as the *find-and-hook-plane* rule set is activated because it does not need to know the object to be clicked. This has the advantage that the hand can be moved to the mouse right at the beginning of the *find-and-hook-plane* rule set, instead of waiting until a plane has been found on the screen. A second consequence is that perceptual processes can continue while the manual processes are busy. During the execution of the manual processes, other planes can be inspected. This is very useful for this task because there are certain criteria ranking the planes, and when a better plane is found it can be selected instead of the first choice.

Fig. 12 shows an example model trace of a novice. The structure of behavior is similar to that shown in the novice behavior in Fig. 3: An instruction is retrieved by a rule, it takes some time to retrieve the instruction, and then the instruction is carried out. Sometimes the execution of the instruction has to wait until the module in question is done with the previous action, which is most notable in the case of the motor actions. A difference with the dual-task model is that the same instruction is retrieved a number of times in a row, once for every step in the instruction. The figure also shows that there is no interleaving between the two instructions yet (represented by thick-bordered and thin-bordered boxes for, respectively, the *find-object* and *click-object* instructions).

Production compilation produces rules that can fire out of sequence, and it allows for an efficient interleaving of perceptual and motor actions (Fig. 13). There is no longer a need for retrieval from declarative memory, because all the instructions have been incorporated in the rules. In this example, perfect dual tasking of the two instructions is achieved. A single rule ini-

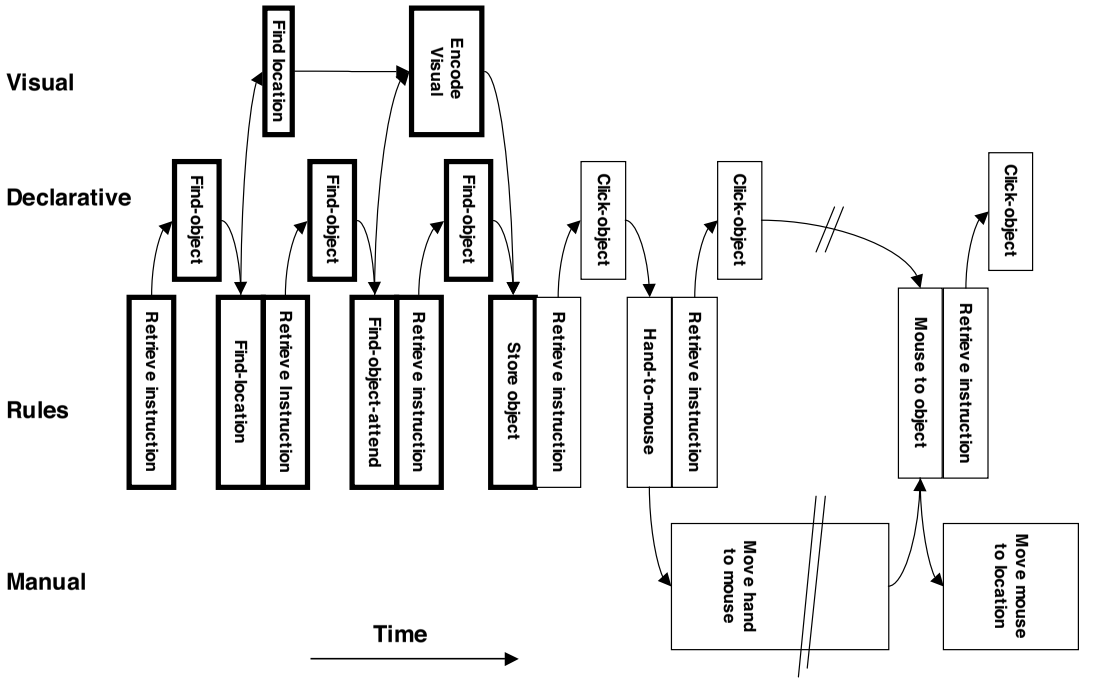

Fig. 12. Partial trace of novice model behavior on selecting a plane in the CMU-ASP task. Steps related to *find-object* are represented by thick-bordered boxes, whereas steps related to *click-object* are thin-bordered boxes.

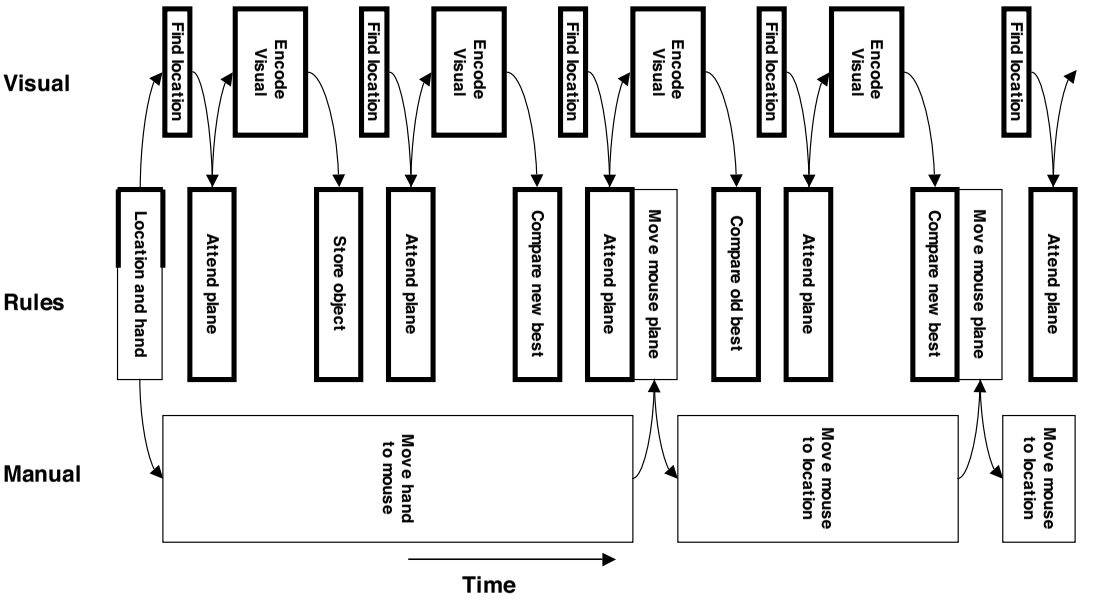

Fig. 13. Partial trace of expert behavior on selecting a plane.

tiates the movement of the hand to the mouse and the perceptual action to find a plane. During the hand movement a second plane is attended to and judged as better than the first plane. Halfway through attending to the third plane the hand movement finishes, enabling a rule that moves the mouse pointer to the best plane up to that moment. However, a fourth plane inspected during this mouse move is better than the plane to which the mouse is moving. Because this destination of the mouse movement is now no longer the target, a second mouse movement is initiated at soon as the first movement is completed.

This example shows that the same principles that produced parallelization in the Schumacher task (Schumacher et al., 2001) produce parallelization of instructions in CMU-ASP. The difference is that, in the CMU-ASP task, learning not only gives a (slight) improvement in speed, but also produces an improvement in the quality of the selection. Fig. 14 shows that this is true for the participants as well as the model, depicting the average point value of the first five planes selected by the model (based on the average of 12 model runs) and by the participants (based on 16 participants). Only the first five planes are used for the analysis, because early in the scenario the planes with the highest values stand out, whereas later in the scenario differences in point values become negligible. Both the model and the participants show a steady increase in selection quality. Initially (during Scenarios 1 and 2) the model does not compare planes, behaving as in Fig. 12, essentially choosing planes randomly. Only later does it compare planes, and it becomes gradually better at it. The participant behavior follows

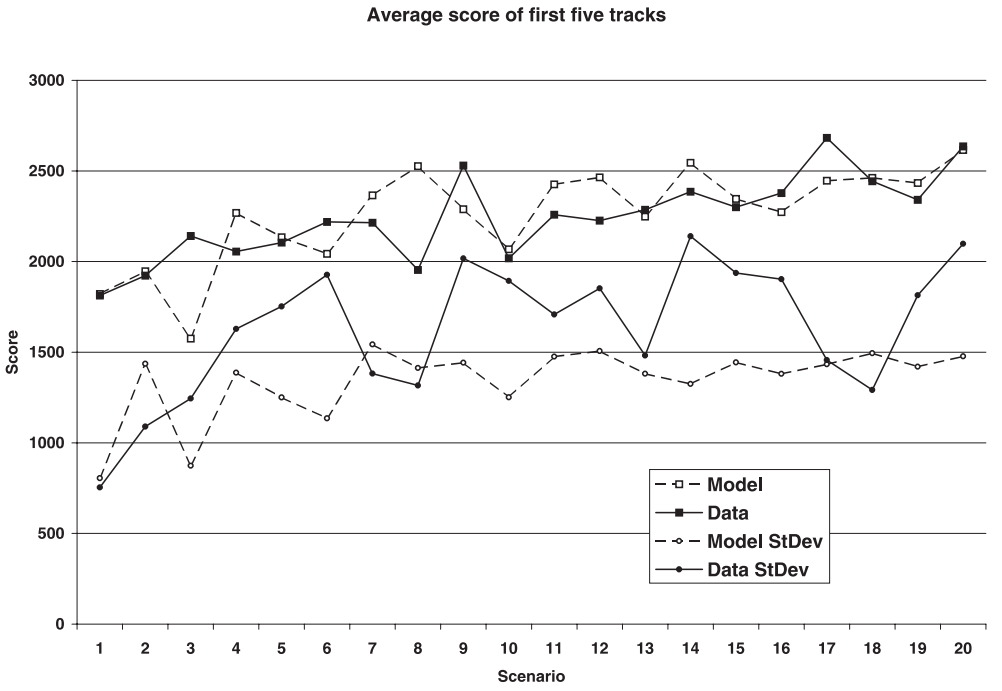

Fig. 14. Average en standard deviation of the point scores for the first five planes.

this trend, both in average scores and variance in the scores, suggesting they also move from random selection to a selection based on comparisons.

3.5. Learning the location of function keys

The *select* rule set contains instructions to press a function key given a label. It first attempts to find the label on the bottom of the screen (*find-label*), maps the location onto the function key (*determine-F-key*), and then presses that key (*press-key*). To explain participants' eventual speed, the model has to learn to press a function key given a label without first looking at the key mappings at the bottom of the screen. The model learns this by integrating several instruction steps, which lead to another case of dual tasking, but with an interesting twist. The two steps that overlap are the *find-label* and *determine-F-key* instructions. *Find-label* will try to find a certain text label (such as "Track Manager") on the screen by simultaneously scanning the display and trying to recall the location. If recall is successful, it will place the location of the recalled position in the visual buffer, directing the visual module to attend that location. The rule learned on the basis of that recall illustrates this:

Learned-Find-Track-Manager-Rule
IF the goal is to select a key for label "Track Manager"
THEN set the location in the visual buffer to coordinates (472,715) and request an attention shift to that location

Normally the model would wait until the attention shift is done and proceed from there. However, the next instruction, *determine-F-key,* only needs the location of the label to determine the right function key. This means that the key can be determined before the attention shift is done. The rule learned from retrieving the instruction is

Learned-Determine-F6-Rule
IF the goal is to select a key
AND the location in the visual buffer has *x*-coordinate 472
THEN set the key to be pressed to F6

The *determine-F-key* instruction needs information that is available halfway through the *find-label* instruction. That means that at some point the rest of the steps to carry out *find-label* are obsolete and can be skipped. The whole process eventually only invokes two production rules, but this takes the model considerable time to learn: It must first compile rules based on the instructions, then it has to incorporate retrieved facts (locations of the labels) into these rules, and only then can it attempt to parallelize them, but they also have to win the competition with the already existing rules in terms of utility. A model with explicit states to sequence the instructions would need a planning system to determine that some steps are no longer needed, but by not having these control states the model discovers this by itself.

3.6. Learning to optimize hand movements

To press one of the function keys, either the left or the right hand can be used. The model has rules to use either the left or the right hand, but in practice the left hand is always on the left side

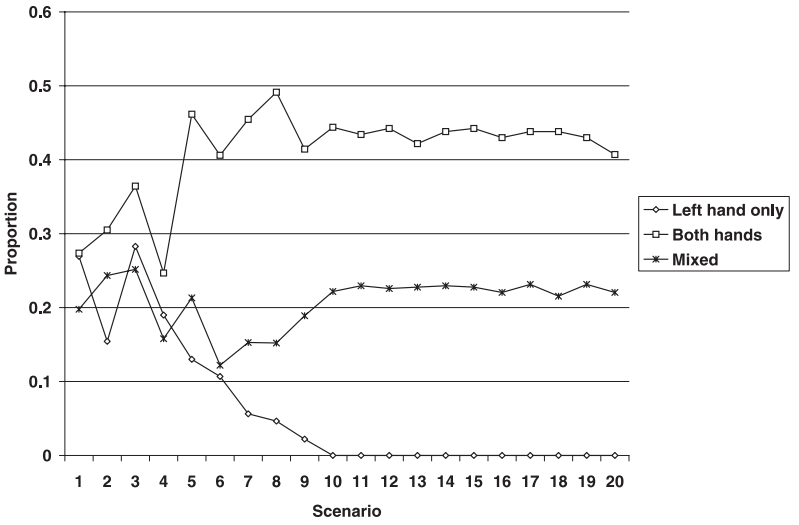

Fig. 15. Proportion of right-hand use for function keys. The function keys on the left side of the keyboard are also included, which are always pressed with the left hand.

of the keyboard, so is always the best choice for function keys on the left side of the keyboard. For the function keys on the right side of the keyboard the model can only use the right hand if the hand is on the keyboard and not on the mouse, otherwise it has to move the hand to the keyboard first. This allows for several different hand-movement styles. The model develops different styles in different model runs, as is depicted in Fig. 15. The figure shows how preference for using the right hand develops over the scenarios. The top curve in the graph comes from a case in which the model prefers to type with both hands, moving the hand back to the keyboard early in the classification process. The bottom curve comes from another run in which the model prefers to do all keying with the left hand and leaves the hand on the mouse. The middle curve is a compromise between the two: The model does the first few keystrokes with the left hand while the right hand is not yet back from the mouse. When the right hand has returned from the mouse, the remaining keystrokes are done with the right hand.

The model develops several styles because none of them are truly superior to the others. Initially the choice of style is random, but the first style to be proceduralized then has an edge over the styles that are not yet proceduralized, making that style even more attractive. In this way the model can end up with a strategy that is optimal only because it is the best practiced.

3.7. Conclusions

In the introduction we noted that many criticisms of rule-based systems focus on two common assumptions: the assumption that instructions are translated into rules right away and the assumption of a strong hierarchical task representation. The two models presented in this article do not rely on these assumptions and yet show a good correspondence with the empirical data. The instruction assumption is replaced by an initial representation of the task that has the form of declarative instructions. By interpreting these instructions, task-specific production

rules are gradually learned. The strong hierarchical task representation assumption has been replaced by a weak two-level hierarchy guided by the minimal control principle, which states that models should use as few control states as possible.

The two models that we have constructed on the basis of these principles can learn their own rules and interact flexibly with both the external world and their own internal resources. The model of the Schumacher et al. (2001) experiments was able to achieve perfect interleaving of the two tasks although relying on only one control state. In the model of CMU-ASP, there is a mix between top-down and bottom-up influences: Instructions that are organized in rule sets are cued by both internal variables and external events. Initially each instruction within a rule set can be considered as a separate control state, but as production compilation creates rules that skip over instructions, eventually each rule set has just one control state. The collapse of control states within a rule set lets the model parallelize steps when possible and also lets new strategies emerge by spontaneous recombination of steps.

Kieras et al. (2000) assumed that skill acquisition goes through five stages with an emphasis on scheduling resources. In contrast to this focus, there are hardly any resource conflicts in the models discussed here. In some cases, conflicts are handled by the serial nature of the architecture: ACT–R processes requests for a resource on a first-come, first-served basis. If this leads to a conflict, it bases its choice on the utility of the competing rules. Another issue that Kieras et al. raised is the transfer of control between different tasks. In the dual-task experiments discussed here, the two tasks are very simple and presented at the same time, so they can be considered as one task. The CMU-ASP model does not involve multiple tasks, just subsequent instructions that are interleaved. When the two tasks are learned separately and each requires its own context, some explicit goal switch is needed (see Salvucci, 2005, this issue, for an example).

Although we addressed the issue of parallel versus serial cognition only in passing, the learning perspective casts a new light on the issue as well. ACT–R assumes that cognition is serial. To achieve true parallelism, very specialized production rules must be learned that address multiple tasks at the same time. Although this is feasible in situations such as the Schumacher et al. (2001) task, it only works in general in overly practiced combinations of tasks, or in cases where interleaving can be achieved easily. In EPIC, on the other hand, parallelism requires a scheduler, and the system must learn an optimal scheduler for the task. In the case of novice combinations of tasks, a general scheduler locks out one or more tasks, effectively enforcing seriality. The issue is therefore transformed into a contrast between ACT–R, which hypothesizes serial processing that must develop parallelism, and EPIC, which hypothesizes that parallel processing is blocked for new tasks, but that this blockage is gradually released. As long as EPIC has no explicit theory on how this blockage is removed, it is hard to assess whether the two theories produce equivalent predictions or not.

It is not clear to what extent our model results can also be achieved by the SOAR architecture (Newell, 1990). In SOAR, operator selection heavily depends on this state, but it is conceivable that in the EPIC-SOAR combination (Chong & Laird, 1997) this dependence could be diminished. Nevertheless, the general skill acquisition theory behind SOAR (Newell & Rosenbloom, 1981) seems to point in the direction of learning massive numbers of highly specialized rules for any combination of control state and input, instead of an attempt to reduce the number of possible control states.

Models of learning new tasks are not interesting only for studies of learning. They also constrain the space of possible expert models. Although the various modeling paradigms for expert models are capable of accurately describing performance, their formalisms allow for many models that do not correspond to reality. Learning models can show what is possible and what is not, and they may even exhibit surprising emerging behavior that the modelers never expected. The addition of the minimal control principle can constrain models even further: It guides model construction to choose the model with the fewest control states. Within ACT–R, the number of control states is easily quantified as the number of possible values of the goal state buffer.

We should note that the approach discussed here is currently limited to situations with a clear set of instructions. Other learning situations, such as learning from demonstration, analogy, and examples, are not yet captured by this framework. Another limitation of this work is that the modeler determines the way the whole task is partitioned into rule sets. Although each rule set must correspond to a single control state, offering some constraint, it would be more desirable if the model could determine its own rule-set boundaries. In future work we will explore how ACT–R models can generate their own instructions on the basis of examples and general knowledge and on partition rule sets when the need for additional control states arises.

Related to this issue is the fact that successful interleaving of tasks is only possible if the model knows the structure of the constraints between the instructions. In the CMU-ASP task, these constraints are fairly trivial: There is no reason why one could not move a hand to the mouse while looking at the screen, or typing a function key while not looking at its label. However, in other situations where instructions must be followed, the constraints can be hidden, especially if the instructions are written out as an explicit recipe, but the function of individual steps is unspecified. If a learner is confronted with such a situation, he or she can only proceduralize the sequence of steps, which will lead to faster execution, but not to increased flexibility. Examples of this problem can be found in programming video recorders (e.g., Gray, 2000) and operating flight management systems in modern airplanes (e.g., Sherry & Polson, 1999). In other words, poor instruction can lead to a situation in which a person has more control states than necessary.

Notes

1. This currently has no associated module, but see Salvucci (2005) for a proposal.

Acknowledgments

This research was supported by ONR grant N00014–04–1–0173 and NASA grant NCC2–1226.

I would like to thank John Anderson, Dan Bothell and Scott Douglass for providing me with the CMU-ASP data, and John Anderson, Stefani Nellen, Dario Salvucci, Pat Langley and an anonymous reviewer for their comments on earlier versions of the manuscript.

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. (1982). Acquisition of cognitive skill. *Psychological Review, 89,* 369–406.

Anderson, J. R. (2005). Human symbol manipulation within an integrated cognitive architecture. *Cognitive Science, 29,* 313–341.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of mind. *Psychological Review, 111,* 1036–1060.

Anderson, J. R., Fincham, J. M., & Douglass, S. (1997). The role of examples and rules in the acquisition of a cognitive skill. *Journal of Experimental Psychology: Learning, Memory and Cognition, 23,* 932–945.

Anderson, J. R., Taatgen, N. A., & Byrne, M. D. (in press). Learning to achieve perfect time sharing: Architectural implications of Hazeltine, Teague, & Ivry (2002). *Journal of Experimental Psychology: Human Perception and Performance.*

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

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

Chong, R. S., & Laird, J. E. (1997). Identifying dual-task executive process knowledge using EPIC-Soar. In M. G. Shafto & P. Langley (Eds.), *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society* (pp. 107–112). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Dreyfus, H. L., & Dreyfus, S. E. (1986). *Mind over machine: The power of human intuition and expertise in the era of the computer.* New York: Free Press.

Fitts, P. M. (1964). Perceptual-motor skill learning. In A.W. Melton (Ed.), *Categories of human learning*. New York: Academic.

Freed, M., Matessa, M., Remington, R., & Vera, A. (2003). How Apex automates CPM-GOMS. In F. Detje, D. Dörner, & H. Schaub (Eds.), *Proceedings of the Fifth International Conference on Cognitive Modeling* (pp. 93–98). Bamberg, Germany: Universitats-Verlag.

Gray, W. D. (2000). The nature and processing of errors in interactive behavior. *Cognitive Science, 24,* 205–248.

Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world performance. *Human–Computer Interaction, 8,* 237–309.

Hazeltine, E., Teague, D., & Ivry, R. B. (2002). Simultaneous dual-task performance reveals parallel response selection after practice. *Journal of Experimental Psychology: Human Perception and Performance 28,* 527–545.

Hodge, K. A., Rothrock, L., Kirlik, A. C., Walker, N., Fisk, A. D., Phipps, D. A., et al. (1995). Trainings for tactical decision making under stress: Towards automatization of component skills. (HAPL-9501). Atlanta: Georgia Institute of Technology, School of Psychology, Human Attention and Performance Laboratory.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general purpose machine learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning II* (pp. 593–623). Los Altos, CA: Kaufmann.

Hornof, A. J., & Kieras, D. E. (1997). Cognitive modeling reveals menu search is both random and systematic. *Proceedings of CHI-97* (pp. 107–114). New York: Association for Computing Machinery.

Kieras, D. E., Meyer, D. E., Ballas, J. A., & Lauber, E. J. (2000). Modern computational perspectives on executive mental processes and cognitive control: Where to from here? In S. Monsell & J. Driver (Eds.), *Control of cognitive processes: Attention and performance XVIII* (pp. 681–712). Cambridge, MA: MIT Press.

Logan, G. D. (1988). Toward an instance theory of automatization. *Psychological Review, 95,* 492–527.

Logan, G. D. (1992). Attention and preattention in theories of automaticity. *American Journal of Psychology, 105,* 317–339.

Meyer, D. E., & Kieras, D. E. (1997). A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review, 104,* 2–65.

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

Newell, A., & Rosenbloom. P. S. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 1–55). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Pashler, H. (1994). Dual-task interference in simple tasks: Data and theory. *Psychological Bulletin, 116*, 220–244.

Rumelhart, D. E., & McClelland, J. (1986). *Parallel distributed programming* (Vol. 1–2). Cambridge, MA: MIT Press.

Salvucci, D. D. (2005). A multitasking general executive for compound continuous tasks. *Cognitive Science, 29,* 457–492.

Schumacher, E. H., Seymour, T. L., Glass, J. M., Fencsik, D. E., Lauber, E. J., Kieras, D. E., et al. (2001). Virtually perfect time sharing in dual-task performance: Uncorking the central cognitive bottleneck. *Psychological Science, 12,* 101–108.

Sherry, L., & Polson, P. G. (1999). Shared models of flight management system vertical guidance. *International Journal of Aviation Psychology, 9,* 139–153.

Taatgen, N. A., & Anderson, J. R. (2002). Why do children learn to say "broke"? A model of learning 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.