SPECIAL SECTION

Production Compilation: A Simple Mechanism to Model Complex Skill Acquisition

Niels A. Taatgen, University of Groningen, Groningen, Netherlands, and **Frank J. Lee,** Rensselaer Polytechnic Institute, Troy, New York

In this article we describe *production compilation,* a mechanism for modeling skill acquisition. Production compilation has been developed within the ACT-Rational (ACT-R; J. R. Anderson, D. Bothell, M. D. Byrne, & C. Lebiere, 2002) cognitive architecture and consists of combining and specializing task-independent procedures into task-specific procedures. The benefit of production compilation for researchers in human factors is that it enables them to test the strengths and weaknesses of their task analyses and user models by allowing them to model the learning trajectory from the main task level and the unit task level down to the keystroke level. We provide an example of this process by developing and describing a model learning a simulated air traffic controller task. Actual or potential applications of this research include the evaluation of user interfaces, the design of systems that support learning, and the building of user models.

INTRODUCTION

In such diverse environments as air traffic control and nuclear power plant operation, researchers in human factors have accumulated extensive empirical knowledge of human performance in complex and dynamic tasks. However, the development of detailed computational models that can explain how people are able to perform and learn such tasks has lagged behind. One reason for this disparity is that theoretical investigations of skill acquisition and empirical investigations of complex and dynamic task performance have existed largely as separate and independent areas of research, with theoretical development of skill acquisition primarily focusing on models of learning simple tasks. Although valuable insights have been gained from studying simple tasks, in order to move toward a more complete theory of skill acquisition, researchers need to develop and test their models against complex and dynamic tasks that are more typical of human learning in the real world.

The goal of this article is to show how the theoretical gap between learning simple tasks and performing complex tasks can be bridged. Our approach offers a new way to conceptualize and validate task analyses and provides insights into the nature of human skill acquisition. We believe our approach can help human factors researchers in developing applications in which learning is an integral aspect of the task. Our approach is to use the ACT-Rational (ACT-R) cognitive architecture, which is grounded in psychological theory, to model learning and performance in complex tasks. The key aspect of the architecture is *production compilation,* a computational account of skill acquisition that combines aspects of theories proposed by Anderson (1982, 1987) and Newell and Rosenbloom (1981). We use production compilation to develop a detailed model of learning in a simulated air traffic control task.

Address correspondence to Niels A. Taatgen, Artificial Intelligence, University of Groningen, Grote Kruisstraat 2/1, 9712 TS Groningen, Netherlands; niels@ai.rug.nl. **HUMAN FACTORS,** Vol. 45, No. 1, Spring 2003, pp. 61–76.

Mechanisms for Skill Acquisition

Skill acquisition has traditionally been viewed as going through three stages: the cognitive, the associative, and the autonomous (Fitts, 1964). In the cognitive stage performance is slow and prone to errors, whereas in the autonomous stage performance is fast and error free. Anderson (1982) posited that these three stages could be understood as a shift from using declarative knowledge to using procedural knowledge. For the cognitive stage, he argued that the knowledge to perform a task is mostly declarative and needs to be interpreted. The process of interpreting declarative knowledge is slow and can lead to errors, especially if the relevant knowledge cannot be retrieved from declarative memory when needed. In the autonomous stage, the knowledge to perform a task is mostly procedural. Procedural knowledge is compiled and therefore fast and free of errors. The associative stage reflects a transitional period during which knowledge is partly declarative and partly procedural.

To model this shift from using declarative knowledge to procedural knowledge, Anderson proposed a learning mechanism he called *knowledge compilation,* which became part of the ACT* cognitive architecture (Anderson, 1983), a predecessor of ACT-R. Knowledge compilation is a computational theory of learning that consists of a two-step process: proceduralization and composition. During proceduralization, domain-specific declarative knowledge is inserted into the procedures, replacing general-purpose knowledge. During composition, multiple-step procedures are collapsed into a single procedure. To further refine procedural knowledge, two more mechanisms are used: generalization (dropping conditions or replacing constants by variables) and specialization (adding conditions or replacing variables by constants). However, Anderson later abandoned knowledge compilation (Anderson & Lebiere, 1998) because of a lack of empirical evidence and problems with what he called "computational misbehavior," which is caused by learned procedural knowledge that brings the system to an endless loop or causes it to abort prematurely. Specifically, there were too many opportunities to learn faulty procedural knowledge.

Newell and Rosenbloom (1981) proposed an alternate theory of skill acquisition called *chunking,* which became an important component of the Soar (Newell, 1990) cognitive architecture. Within Soar, learning is a result of impasses and subsequent subgoaling. Whenever Soar reaches an impasse (i.e., there is no applicable procedure or no way to resolve the choice between multiple procedures), it automatically creates a subgoal to resolve the impasse. When a subgoal succeeds and the impasse is resolved, new procedural knowledge is created that summarizes all the processing required to achieve that subgoal. If Soar encounters a similar impasse in the future, it can simply apply the newly learned procedure to solve it in a single step.

The strength (and perhaps also weakness) of chunking in Soar is that it constrains the modeler to achieve all learning through a single unified learning mechanism. Although this is highly desirable, it is not without problems. Because of the unpredictable nature of the generalization process of chunking, it sometimes produced procedures that were too general. In addition, both Soar and ACT* suffered from their respective learning mechanisms, producing increasingly larger macroprocedures, which become computationally intractable to match (Tambe, Newell, & Rosenbloom, 1990) as well as too powerful and unconstrained.

Production Compilation

Production compilation was developed by Taatgen and Anderson (2002), and is currently incorporated in the latest version of the ACT-R theory (Anderson, Bothell, Byrne, & Lebiere, 2002). It relies on the same theoretical commitments as ACT-R, such as the distinction between two types of memory (declarative and procedural) and the view that declarative knowledge is represented as semantic networks and procedural knowledge as production rules (which we will refer to as *rules* henceforth for the sake of brevity). Production compilation combines two mechanisms from ACT* – proceduralization and composition – into a single mechanism. In production composition, the composition of two rules into a new single rule is accompanied by a proceduralization step in which a condition is eliminated. Given that ACT-R's rules can have, at most,

only one retrieval from declarative memory, production compilation guarantees that the new rule also has, at most, one retrieval from declarative memory. The following example illustrates production compilation.

Consider the following three production rules to find the sum of three numbers:

Rule 1:	IF	the goal is to add three numbers,
	THEN	send a retrieval request to declarative memory for the sum of the first two numbers.
Rule 2:	IF	the goal is to add three numbers AND the sum of the first two numbers is retrieved,
	THEN	send a retrieval request to declarative memory for the sum that has just been retrieved and the third number.
Rule 3:	IF	the goal is to add three numbers AND the sum of the first two and the third number is retrieved,
	THEN	the answer is the retrieved sum.

In the ACT* and Soar learning mechanisms, one general rule could be learned from these three rules that could do any three-number addition in a single cognitive step. From a psychological perspective this is not a desirable learning outcome, given that people generally cannot do these types of additions in one step, even with extensive experience doing them.

In production compilation, however, new rules are learned by specializing and combining rules that fire in sequence while rigorously maintaining the hard constraint of performing only one retrieval from declarative memory. This is accomplished by eliminating the retrieval request in the first rule and the retrieval condition in the second rule. Suppose in the example just given that the three numbers that are being added are 1, 2, and 3. This would produce two new rules: a combination of Rules 1 and 2 and a combination of Rules 2 and 3.

Rule 1&2:	IF	the goal is to add 1, 2, and a third number,
	THEN	send a retrieval request to declarative memory for the sum of 3 and the third number.
Rule 2&3:	IF	the goal is to add three numbers and the third number is 3, AND the sum of the first two numbers is retrieved and is equal to 3,
	THEN	the answer is 6.

Each of these two rules can be combined with one of the original three rules to form a new rule that combines all three rules:

Rule 1&2&3:	IF	the goal is to add 1, 2, and 3,
	THEN	the answer is 6.

Compared with the original three rules, this rule is very specialized: It works only for the specified numbers 1, 2, and 3. One implication of production compilation is that people learn new rules only if they are more specific than the rules they already know, and therefore they learn rules only for situations that occur often. For example, people probably learn the sum of 1, 2, and 3 but not that of 9, 3, and 4.

Production compilation is very suitable to model skill acquisition. Consistent with the assumptions in the ACT-R theory, it also assumes that a skill is originally represented declaratively, which may be incomplete and may require some additional experience (episodic declarative knowledge). Retrieving from declarative memory is a very slow process because only one memory can be retrieved at a time. Production compilation speeds up this process by producing task-specific procedural knowledge (i.e., production rules). The following example is from the air traffic controller task that we will discuss in detail later; briefly, one aspect of this task is that planes have to be landed, and in order to do this one's visual attention has to be directed to the part of the screen that lists the planes that can be landed. Initially this knowledge is declarative: There are instructions in declarative memory that in order to land a plane, one has to attend to Hold Level 1 (a specific area on the screen) and that the location of Hold Level 1 is on the bottom left of the screen. In order to interpret these facts one needs the following three rules:

Retrieve instruction:
IF you have to do a certain task,
THEN send a retrieval request to declarative memory for the next instruction for this task.

Move attention:
IF you have to do a task AND an instruction has been retrieved to move attention to a certain place,
THEN send a retrieval request to declarative memory for the location of this place.

Move to location:
IF you have to do a task AND a location has been retrieved from declarative memory,
THEN issue a motor command to the visual system to move the eyes to that location.

These three rules are general procedures and may be used in any task, but when production compilation combines them with the declarative instruction specific to the air traffic controller task, it produces three task-specific rules. The first two rules are again combinations of pairs of the original rules:

Instruction & attention:
IF you have to land a plane,
THEN send a retrieval request to declarative memory for the location of Hold Level 1.

Attention & location:
IF you have to do a task AND an instruction has been retrieved to move attention to Hold Level 1,
THEN issue a motor command to the visual system to move the eyes to the bottom left of the screen.

Combining either of these two rules with the rule from the original set (i.e., combining "instruction & attention" with "move to location" or "retrieve instruction" with "attention & location") produces the following task-specific rule for landing a plane:

All three:
IF you have to land a plane,
THEN issue a motor command to the visual system to move the eyes to the bottom left of the screen.

This example shows how new procedural knowledge of a task can be learned in such a way that it considerably speeds up performance. Another important aspect of skill acquisition is that experience has to be incorporated into performance. In the air traffic controller task, for example, only certain types of planes may be landed on certain runways. Suppose you had a successful experience of landing a DC-10 on the short runway. This experience is then stored in declarative memory and can later be retrieved when you are faced with a similar situation:

Look for experience:
IF you do a task, and you have to decide upon some action,
THEN send a retrieval request to declarative memory for a successful experience similar to the current situation.

Decide upon experience:
IF you do a task, and you have to decide upon some action, AND a successful experience has been retrieved similar to the current situation,
THEN take the same action as taken in the retrieved experience.

Learned rule:
IF you have to land a DC-10,
THEN land it on the short runway.

The two original rules ("look for experience" and "decide upon experience") can be used in any context, but when combined with the prior experience of landing a DC-10 they produce a task-specific rule. Note that this may lead to generalization and also to overly generalized rules. Indeed, in the real task, DC-10s can be landed on the short runway only sometimes, depending on the current weather and runway conditions. However, the error of overgeneralization results from following the strategy of using old experiences and does not reflect an inherent weakness in production compilation.

Although production compilation is similar to the older mechanisms on which it builds, it does not suffer from the problems of those mechanisms. First, it produces only compact production rules using a single mechanism, making it very constrained. Second, it will not produce production rules that lead to "computational misbehavior," making it very robust. Additionally, because it is embedded in the ACT-R cognitive architecture, it is also open to empirical testing.

By using production compilation to develop computational models of complex tasks, human factors researchers can better conceptualize learning in complex tasks and thereby offer better insight into the processes underlying skill acquisition in those tasks. Production compilation has already been successfully used to model inflection of the English past tense (Taatgen & Anderson, 2002), the German plural (Taatgen, 2001), and strategy development in the balanced-beam task (van Rijn, van Someren, & van der Maas, in press). In the remainder of this article we will explore how production compilation can be used to model a complex task.

THE AIR TRAFFIC CONTROLLER TASK

The Kanfer-Ackerman air traffic controller (KA-ATC) task (Ackerman, 1988; Ackerman & Kanfer, 1994) is a simplified simulation of an air traffic controller task. It captures certain dynamic aspects of real air traffic control, such as planes losing fuel in real time and random changes in the weather, but it is simple enough to be tractable to study. Figure 1 displays a prototypical layout of the KA-ATC task.

In the KA-ATC task, participants can accept a plane from the queue into an open hold position, move a plane among the three hold levels, and land a plane on a runway by using four keys: up arrow, down arrow, F1, and Enter. They can move the cursor up and down the hold positions and the runways using the up- and down-arrow keys; accept a plane from the queue into an open hold position using the F1 key; and select a plane in the hold, place a selected plane in an open hold position (either from the queue or from another hold position), or land a plane on a runway using the Enter key.

Strict rules govern landing planes in this task. Planes can land only from Hold Level 1 onto an unoccupied runway consistent with the current wind direction, and ground conditions and wind speed determine the runway length required by different plane types. Participants receive 50 points for landing a plane, are penalized 100 points for allowing a plane's fuel level to fall to 0 min in the hold, and are penalized

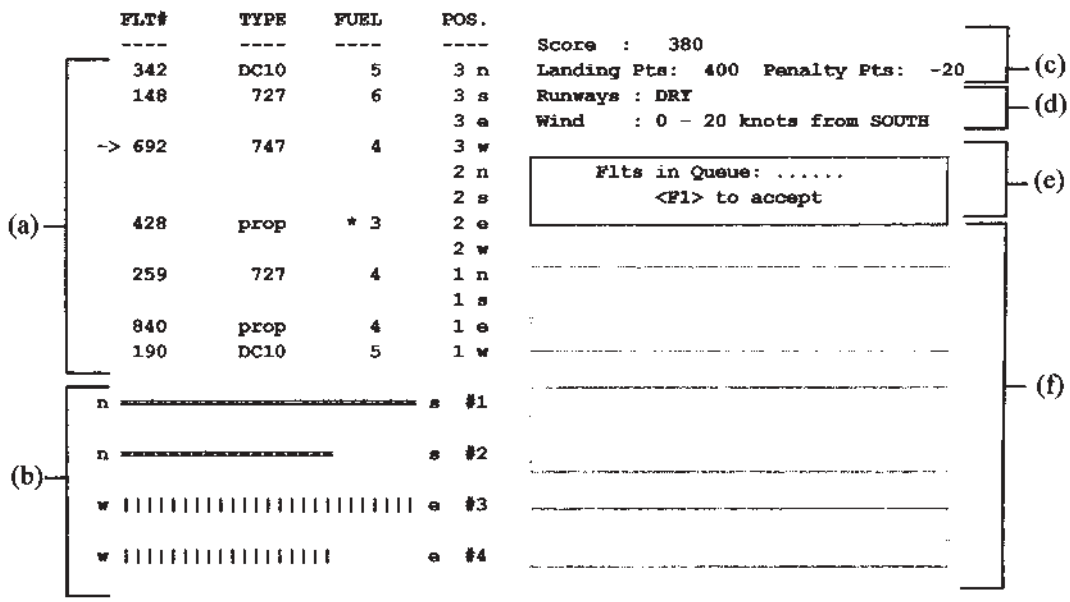


Figure 1. The KA-ATC task. The display is composed of the following items: (a) 12 hold positions (spaces that can hold airplanes), either empty or containing a plane; (b) four runways, two of each length (short and long) and two of each direction (north-south and west-east); (c) current score; (d) current runway and weather conditions; (e) the queue of planes waiting to be entered into hold positions, each dot representing a plane; and (f) three message windows. The 12 hold positions are divided into three levels roughly corresponding to altitude.

10 points for violating a rule. Planes are added to the queue approximately every 7 s, and it takes 15 s for a plane to clear a runway. Once planes enter the hold position from the queue, they have 4 to 6 min of fuel and begin to lose fuel in real time.

Previous Efforts to Model the KA-ATC Task

Lee and Anderson (2000) developed a model of expert performance in the KA-ATC task using ACT-R perceptual motor (ACT-R/PM; Byrne & Anderson, 1998) cognitive architecture. In their model, Lee and Anderson showed that expert performance in the KA-ATC task required a substantial degree of parallelism among cognition, perception, and action. Indeed, as people became skilled in the KA-ATC task, their performance was largely limited by the constraints on the motor system. Lee and Anderson (2000) modeled only expert performance in the KA-ATC task and did not model learning, which is the focus of the present model.

In addition, Taatgen (2002) developed an ACT-R model of the impact of individual differences on performance and learning speed. By manipulating ACT-R parameters corresponding to working memory capacity, speed of production compilation, and psychomotor speed, he found the same pattern of correlations between individual differences and performance as Ackerman (1988) found in his empirical analysis. However, the model only approximated the interaction between the participant and the task, could not achieve the same levels of performance as the human participants, and was unable able to model human performance at the level of unit tasks and keystrokes. The model we present in this paper is an extension and revision of this model. In the current model, we account for perceptual and motor interaction with the interface of the air traffic controller task and the learning and performance exhibited by human participants.

THE MODEL

The model consists of two parts. The first part is a declarative representation of what the participant has to do in the experiment. This representation is based on the instructions that the participants receive for the task and on the task analysis made by Lee and Anderson (2001). The second part is a set of domain-general production rules, among which are rules to attend to regions on the screen, to press sequences of keys, to change between goals, and to repeat certain actions until a criterion is satisfied. The declarative representation we use is just one of many possible interpretations of the instructions; other interpretations are possible. That is, because our interpretation of the instructions stems from our specific task analysis, different analyses of the task may lead to different interpretations. However, our analysis of the task enables our model to learn the task as well as people do, which is one way to evaluate the sufficiency of a task analysis.

Our model of the KA-ATC task is based on the ACT-R cognitive architecture (Anderson et al., 2002). In particular, we use Version 5 of ACT-R, which incorporates many new theoretical elements, including a perceptual-motor extension. We will first give an overview of the ACT-R cognitive architecture followed by a description of our model. We then proceed to discuss in more detail those aspects of ACT-R that are particularly important for the present model.

Overview of the ACT-R Architecture

The theoretical foundation of the ACT-R architecture is the rational analysis of human cognition (Anderson, 1990). According to rational analysis, each component of the cognitive system is optimized with respect to the demands from the environment, given its computational limitations. The main components of ACT-R are the two memory systems: the declarative (fact) memory and the procedural (skill) memory. ACT-R is a hybrid architecture in that it has both symbolic and subsymbolic layers. Items in declarative memory, called *chunks,* have different levels of activation to reflect their use: chunks that have been used recently or chunks that are used frequently have high levels of activation. Activation of a chunk decays over time if it is not used. In addition, chunks are by themselves inert; they require production rules for their application. In order to use a chunk, a production rule has to retrieve it from declarative memory and another production rule has to do something with it.

Because ACT-R is a goal-driven theory, chunks are always retrieved to achieve some goal. In the context of the KA-ATC task, there are several goals. Although only one goal can be active at any one time, a model may use elaborate strategies to switch between goals. One goal may be to land a plane, for which it may be necessary to hand control over to a lower-level goal (e.g., a goal to move the arrow on the screen to the desired plane).

The behavior of production rules is also governed by the principle of rational analysis. Each production rule maintains a set of parameters that is used to calculate its expected outcome. The expected outcome of a rule is derived from the estimated cost (in time) and probability of reaching the goal if that rule is chosen. ACT-R's learning mechanisms constantly update these estimates based on experience. If multiple production rules are applicable for a goal, the rule with the highest expected outcome is selected.

In both declarative and procedural memory, a selection is made on the basis of some evaluation, either activation or expected outcome. However, this process is noisy, so although the item with the highest value has the greatest probability of being selected, other items do occasionally get opportunities to be selected. Although having noise may produce errors or suboptimal behavior, it also allows the system to explore knowledge and strategies that are still evolving. In addition to learning through fine-tuning the activations of chunks and expected outcomes of production rules, ACT-R can also learn new chunks and production rules. New chunks are learned automatically: Each time a goal is completed, it is added to declarative memory. If an identical chunk is already present in memory, both chunks are merged and their activation values are combined. Chunks are also acquired through perception, such as when information on the screen is encoded into declarative memory. New production rules are learned through production compilation.

The current ACT-R theory also offers a perceptual-motor interface to the architecture that allows for accurate predictions of attentional shifts and motor actions. It enables the model to interact directly with the experimental software, ensuring that the model uses the same methods that the participants use to interact with the external task. Although the central core of ACT-R is serial, the different subsystems (the visual system, declarative memory, the motor system) can act asynchronously. For example, when declarative memory is busy with a retrieval, a production rule not acting on declarative memory may issue a command to move visual attention to a new location.

Task Analysis

Figure 2 illustrates Lee and Anderson's (2001) decompositional task-analysis of the KA-ATC task. The task analysis is based on Card, Moran, and Newell's (1983) method of unit-task analysis, in which a task is decomposed into increasingly specific goals, all the way down to the keystroke level. Lee and Anderson's (2001) analysis of the keystroke level includes not only keystrokes but also changes in visual attention, which were then compared with eye movement data. As can be seen in Figure 2, the KA-ATC task can be decomposed into three unit tasks: (a) moving a plane between hold levels, (b) landing a plane on a runway, and (c) getting a plane from the queue into a hold position. As Figure 2 further illustrates, each unit task can be decomposed into a number of functional-level goals. For instance, the unit task of landing a plane involves (a) finding a plane to land, (b) moving to the plane, (c) selecting the plane, (d) finding a runway to land on, (e) moving to the desired runway, and (f) landing the plane. Each of these functional-level goals involves a number of keystroke-level goals, including a sequence of shifts of attention across the screen, encoding of information on the screen, and a keystroke to effect the desired action. We illustrate only the land unit task because both the move and queue unit tasks are nearly identical in structure to the land unit task but are less complicated.

Description of the Model

As indicated in the Introduction, the basis for the model is the idea that instructions are represented in declarative memory and need to be retrieved and interpreted. The production rules that interpret the declarative instructions are not task specific and can be used for other tasks as well. The declarative representation that is used in our model is a mixture of ideas expressed by Taatgen (1999) and by Anderson (2000).

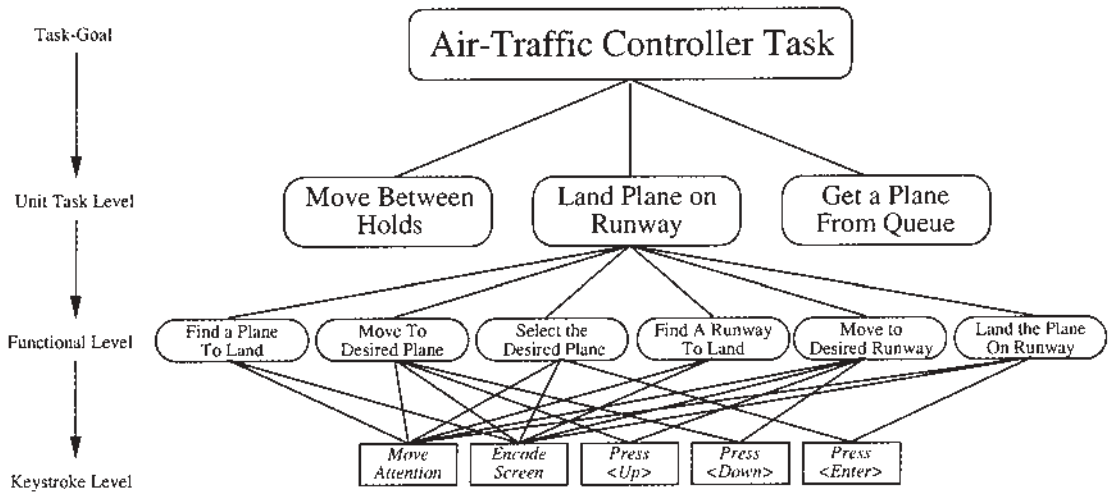


Figure 2. A hierarchical task decomposition of the KA-ATC task. Only the "land plane on runway" unit task is expanded to the functional level. Adapted from *Cognitive Psychology,* Vol. 42, No. 2, Lee & Anderson, "Does learning a complex task have to be complex? A study in learning decomposition," pp. 267–316, copyright 2001, with permission from Elsevier.

It is organized in two levels: The top level consists of goals, and the bottom level consists of individual steps that have to be taken to complete a goal. The individual steps of a goal are organized sequentially. For example, to land a plane, one must first look at Hold Level 1 to see if it contains any planes, then look at the wind direction to determine which runways are currently usable, and finally look at the two usable runways to determine their availability (i.e., occupancy). At that point a decision has to be made about what to do next, so a switch is made to one of four possible goals: If both runways are occupied or there is nothing in Hold Level 1, a goal is selected to move a plane from Hold Level 2 to Hold Level 1. If only the short runway is free, a goal that searches for a non-747 in Hold Level 1 is selected.

In addition to declarative knowledge representing the instructions, there is also declarative knowledge that controls the switching between goals. In general, there is declarative knowledge that specifies what to do when a goal is completed successfully, what to do when a goal fails, and what to do at explicit choice points in a goal. For example, there is a chunk specifying that if the land goal fails (which happens when no plane is to be found in Hold Level 1), the model should switch to the move goal. Another chunk specifies that once the land goal has arrived at the do-landing action, and its argument is that both runways are free, the model should switch to the goal depicted in Figure 3 as "land long free short free." Figure 3 shows the full goal structure of the model. The locations of objects on the screen (e.g., the fact that Hold Level 3 is in the top-left corner of the screen) are also represented and stored in declarative memory.

The procedural knowledge contains no task-specific rules. It has rules to implement the goal-switching scheme that we described previously and can perform the operations in Table 1. Table 2 is an example of the detailed instructions for implementing "land any plane on long rw" from Figure 3. This goal is executed when planes are available to land and the long runway is free. Each instruction has an action slot (action), two argument slots (arg1 and arg2), and a slot that refers to the previous instruction (prev). The content of the action slot refers to one of the actions that are implemented by the rules in Table 1. The argument slots are used to pass on arguments to the action, sometimes with literal values, sometimes with references to other declarative knowledge (for example, screen locations), and sometimes with references to information stored in the goal (loc1

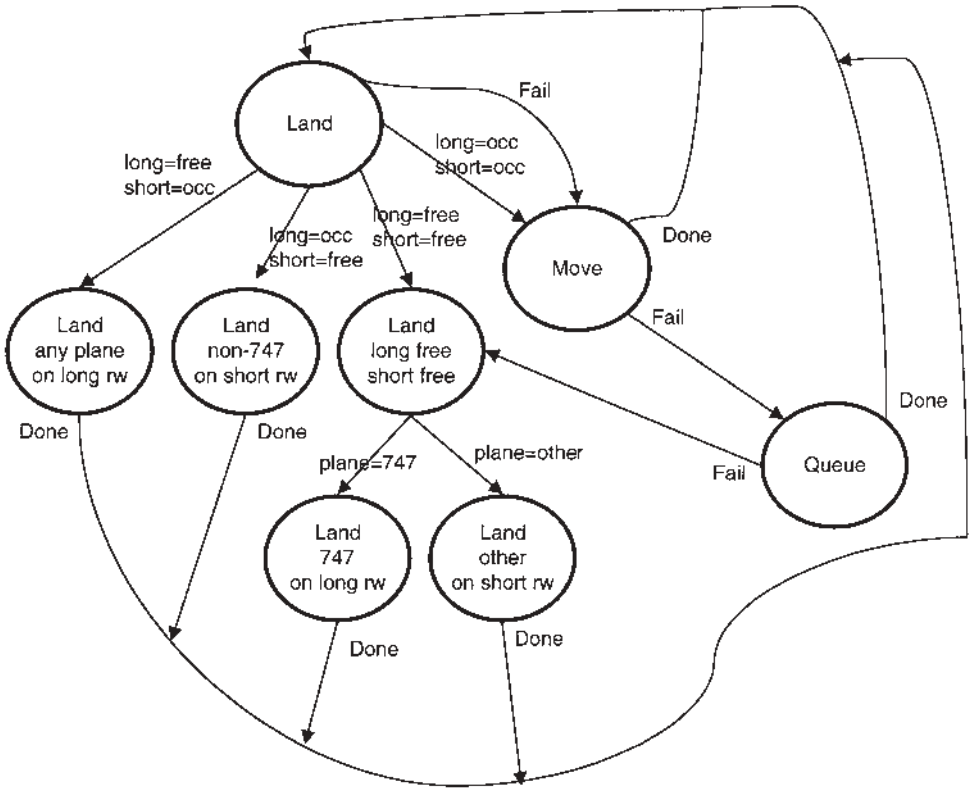


Figure 3. Structure of the goals used by the model. Each circle represents a goal used by the model; the arrows show conditions on which a change of goal is made. "Long = free" means "long runway in the current wind direction is free," "plane = 747" means "currently attended plane is a 747," and so forth; rw = runway.

TABLE 1: Operations That Can Be Performed by Task-Independent Procedural Knowledge

Operation	Description
Scan Scan seek Scan empty	The scan operations move visual attention to a certain region on the screen, focus on an object in that region, and perceive that object. Scan seek will focus only on objects that have not been recently attended, and scan empty will focus on empty space instead of text (necessary to find empty slots in the hold levels).
Remember loc Remember string Remember status	The remember operations store information from the current visual location or object in the goal. Remember loc stores the current visual location. Remember string parses the currently attended word and stores it in the goal. Remember status parses a runway string and stores its status (free or occupied) in the goal.
Press enter Press F1	Operations to press the enter and F1 keys, respectively.
Move to loc	Operation to move the cursor to a specified position on the screen by repeated pressing of the arrow keys.
Compare restart	Compare whether the two arguments of the action are equal. If this is the case, restart the present goal; if not, continue. This operation is used in combination with scan-seek to find a non-747 in Hold Level 1.

TABLE 2: Example of Declarative Instructions for Landing an Arbitrary Plane on the Long Runway

Land-fo-1 action scan arg1 hold-level-1 arg2 plane-type prev start	Land-fo-5 action scan arg1 wind-direction arg2 none prev land-fo-4	Land-fo-9 action move-to-loc arg1 var1 arg2 right-column prev land-fo-8
Land-fo-2 action remember-loc arg1 loc 1 arg2 none prev land-fo-1	Land-fo-6 action remember-string arg1 loc 1 arg2 none prev land-fo-5	Land-fo-10 action press-enter arg1 none arg2 none prev land-fo-9
Land-fo-3 action move-to-loc arg1 var 1 arg2 left-column prev land-fo-2	Land-fo-7 action scan arg1 runway-long arg2 var1 prev land-fo-6	Land-fo-11 action done arg1 none arg2 none prev land-fo-10
Land-fo-4 action press-enter arg1 none arg2 none prev land-fo-3	Land-fo-8 action remember-loc arg1 loc 1 arg2 none prev land-fo-7	Gc53 task do-landing-fo type done newtask land

Note: Land-fo refers to "long runway free and short runway occupied" and corresponds to "land any plane on long rw" in Figure 3. First an arbitrary plane in Hold Level 1 is attended (Instructions 1 and 2). Then the arrow is moved to that plane and enter is pressed (Instructions 3 and 4). In the next steps, the wind direction is checked because it determines which of the two long runways has to be used (Instructions 5 and 6). Finally, the appropriate runway can be attended (Instructions 7 and 8) and the arrow can be moved to it, after which the final enter is pushed (Instructions 9 and 10). The gc53 instruction specifies that after the goal do-landing-fo has successfully been completed, the new goal will be land.

means information is stored in the goal; var1 means this information is retrieved again). Table 2 also shows an example of a chunk that specifies a goal switch. It states that once the example goal is completed, the model should proceed with the land goal.

As the model begins, the instructions are retrieved one by one and carried out by the appropriate production rules. Once the model gains sufficient experience, production compilation produces new rules that create shortcuts through these instructions, similar to the examples described in the Introduction. For example, a rule that is learned from the example in Table 2 summarizes the action in the two steps, land-fo-4 and land-fo-5:

IF the task is land-fo and step land-fo-3 is completed,
THEN issue a motor command to the manual system to press enter AND
issue a motor command to the visual system to move the eyes to (200,50) AND
note that step land-fo-5 is completed.

This production issues a command to press a key and to attend to a location on the screen at the same time, without any intervening declarative memory retrieval. In doing so, it saves time by parallelizing perceptual and motor actions and by eliminating slow declarative memory access.

It has to be noted that the particular strategy that we use in our model is not the only possible strategy. For example, whereas the strategy that we implemented gets only one plane from the queue before checking whether it is possible to land it or move it, an alternate strategy is to bring many planes from the queue in sequence, speeding up the queuing process but potentially missing landing opportunities. These and other strategies may be modeled by extending the definition of what constitutes a unit task and the priorities between them. For instance, the queue unit task could be changed to getting a set of planes from the queue instead of one. In addition, some participants may still be learning the exact nature of the unit tasks and their priorities and will switch strategies later on (John & Lallement, 1997).

One important feature of the model is that it interacts with the experimental software (the

version used by Lee & Anderson, 2001, for their experiments) by issuing perceptual and motor commands. Another important feature of the model is that it runs with all the parameters in ACT-R set to their default values. ACT-R has been criticized in the past for having too many free parameters, limiting the scientific value of its predictions. As a consequence, more care is taken within the ACT-R community to be consistent and principled in setting the parameters. By using the recommended default values for the parameters that have been established in other studies, our model exemplifies this effort.

COMPARISON

To judge the accuracy of the model, we compared the model predictions with data made available in Ackerman and Kanfer (1994, Study 2), as reported in Ackerman (1988). The data from Study 2 were from 65 college undergraduates who completed 18 trials of the KA-ATC task, with each trial lasting 10 min. For our model comparison, we used only Trials 1 through 10. We compared the model's predictions and the data at three levels of detail: (a) overall performance, (b) unit task level performance, and finally (c) keystroke level performance. In order to get the model's predictions, we ran the model five times for 10 trials and averaged the results. All ACT-R parameters were set to their default values.

Overall Performance

As a measure of overall performance, we took the number of planes that were landed within a 10-min trial. Figure 4 shows the participant data and the predictions of the model. Initially the model outperformed the participants, but after a few trials the correspondence between the model and the data was quite good (r^2 = .97 over all 10 trials). The initial advantage of the model is probably attributable to the fact that it has all the declarative knowledge it needs to do the task, whereas participants must learn some of this information while they are learning to do the task initially. For example, Lee and Anderson (2001) attributed their participants' initial speedup mostly to the fact that they started to learn where all the visual information on the screen was located and thus could avoid unnecessary eye movements. The present model initially has all the visual locations in declarative memory and does not need to search for them. Also, the model starts out with a reasonably efficient strategy in declarative memory, which may be true for some participants but certainly not for all of them.

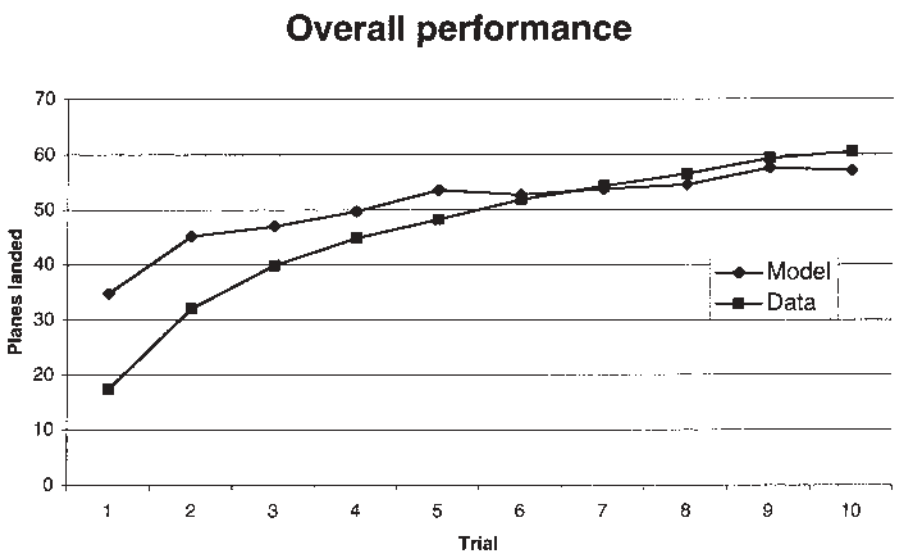


Figure 4. Number of planes landed by the participants (data) and the model for each 10-min trial.

Performance at the Unit Task Level

As has been indicated in the task analysis, three unit tasks can be identified: landing a plane, moving a plane between hold levels, and getting a plane from the queue. Figure 5 shows the empirical results along with the model's predictions for the time it takes to complete each of the three unit tasks. As can be seen, although the fit between the model and the data for the land unit task is good (r^2 = .98), the model is generally slower than the participants in the experiment for the remaining two unit tasks. This is especially true for the move unit task, for which the model is much slower, except in the first trial. An explanation for this can be found in our choice of strategy for the model. The strategy is to check whether a landing is possible before attempting a move unit task, and hence the model will never do consecutive move unit tasks without first checking the wind direction and the occupancy of the runways. For the queue unit task, apart from the initial trial, the participants are also faster than the model (r^2 = .91). This may again be attributable to the model doing a queue unit only task after checking the unavailability of other options (i.e., land and move unit tasks). Nevertheless, the match is quite reasonable, especially considering that we are using the default ACT-R parameters.

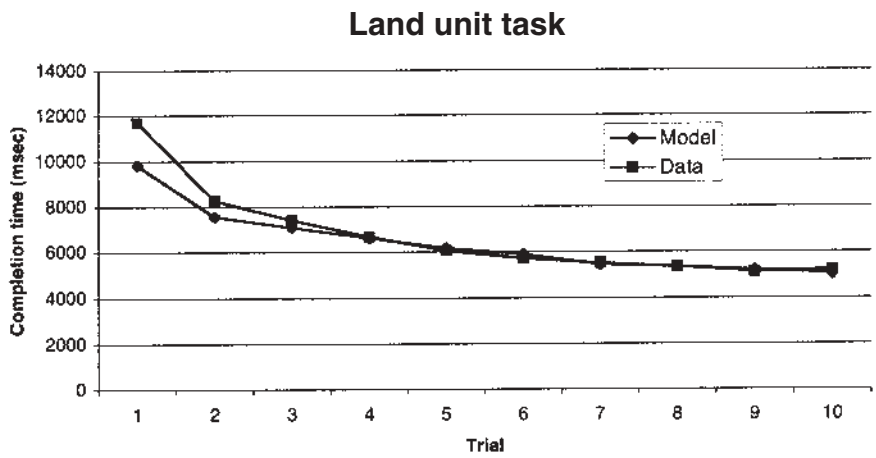
Performance at the Functional and Keystroke Levels

With respect to analyzing performance at the functional and keystroke levels, we limit our discussion to the land unit task because it is the most complicated of the unit tasks. In the land unit task, the decision about which plane to land on what runway has to be made while taking into account the weather conditions. The move and queue unit tasks are much more straightforward in that they involve no critical decisions. As discussed previously, there are six functional-level goals for the land unit task: find plane, move to plane, select plane, find runway, move to runway, and select runway. Because of the absence of eye-movement data in the Ackerman study (Ackerman, 1988; Ackerman & Kanfer, 1994, Study 2), the comparison between model and data will involve keystrokes only. Most of the functional-level goals are associated with only one keystroke, except for the move-to-plane and move-to-runway goals. We will therefore analyze the functional level and the keystroke level together.

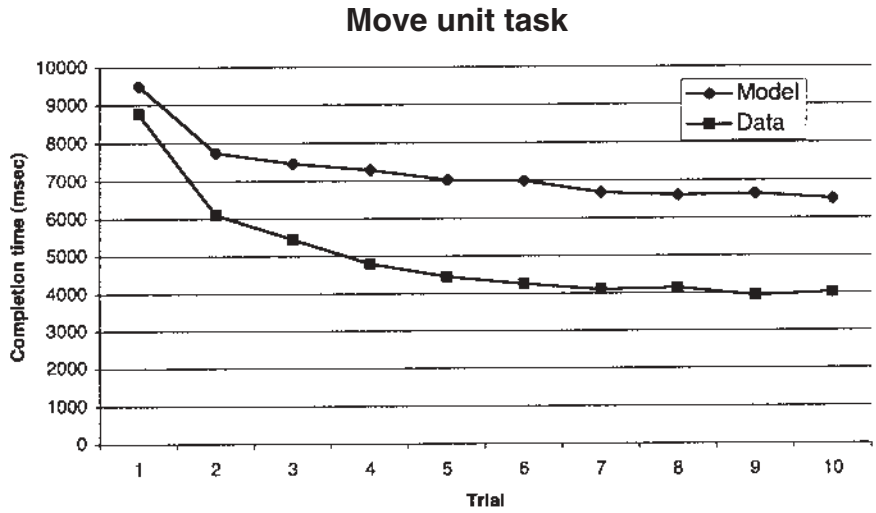
Find plane time is the time between the end of the previous unit task and the first keystroke. *Move to plane time* is the average time for each arrow key press moving to the target plane. *Select plane time* is the time between the final arrow key press used to arrive at the plane and the enter key press to select the plane. *Find runway time* is the average time between the enter key press that selected the plane and the first arrow key press toward the runway. *Move to runway time* is again the average time for each arrow key press moving to the runway. Finally, *select runway time* is the time between the final arrow key press used to arrive at the runway and the enter key press to land on the runway.

Figure 6 shows the time to complete these keystroke-level goals for both the data and the model (r^2 = .87). If we compare the predictions of the model and the performance of the participants, the qualitative pattern is the same. That is, the keystroke times are in the right order, with the slowest keystrokes for the model also being the slowest keystrokes for the participants. In addition, the learning patterns are very similar. The main discrepancy between the model and the data is that the model is slower for the more "cognitive" keystrokes, such as finding and selecting a plane, and is slightly faster for the move to plane and move to runway keystrokes. The predictions for finding and selecting the runway are accurate. One explanation for the discrepancy is that the model rarely makes unnecessary keystrokes, whereas participants do. For example, if a participant moves the arrow a couple of keystrokes toward a certain plane and then decides to select another plane, the finding of the second plane is counted as being part of "move to plane." In this sense, the "find a plane" latencies are diluted into other categories of keystrokes, mainly "move to plane" keystrokes, for which the model is actually faster than the participants. We again believe that this simply reflects the fact that the model starts out with an accurate, complete declarative representation of what to do in the task, whereas participants must sometimes learn while they are doing the task.

(a)



(b)



(c)

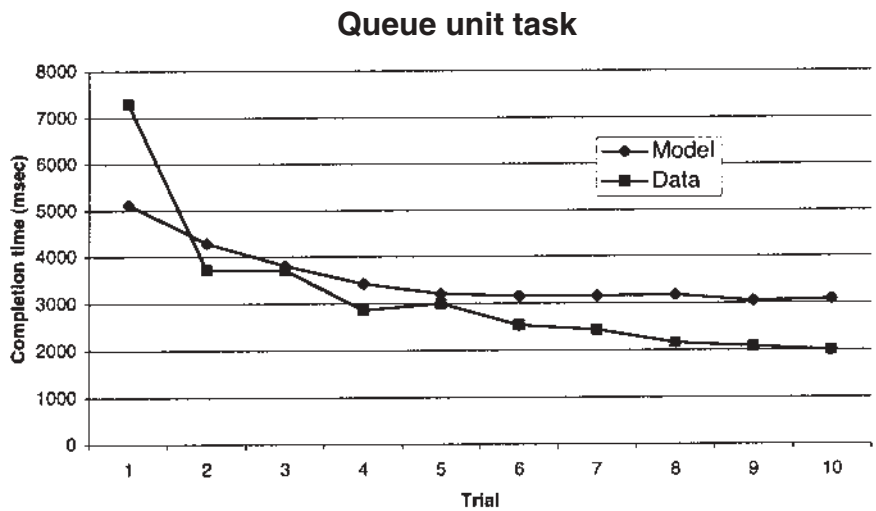


Figure 5. Unit task performance by the participants (data) and the model for (a) land unit task, (b) move unit task, and (c) queue unit task.

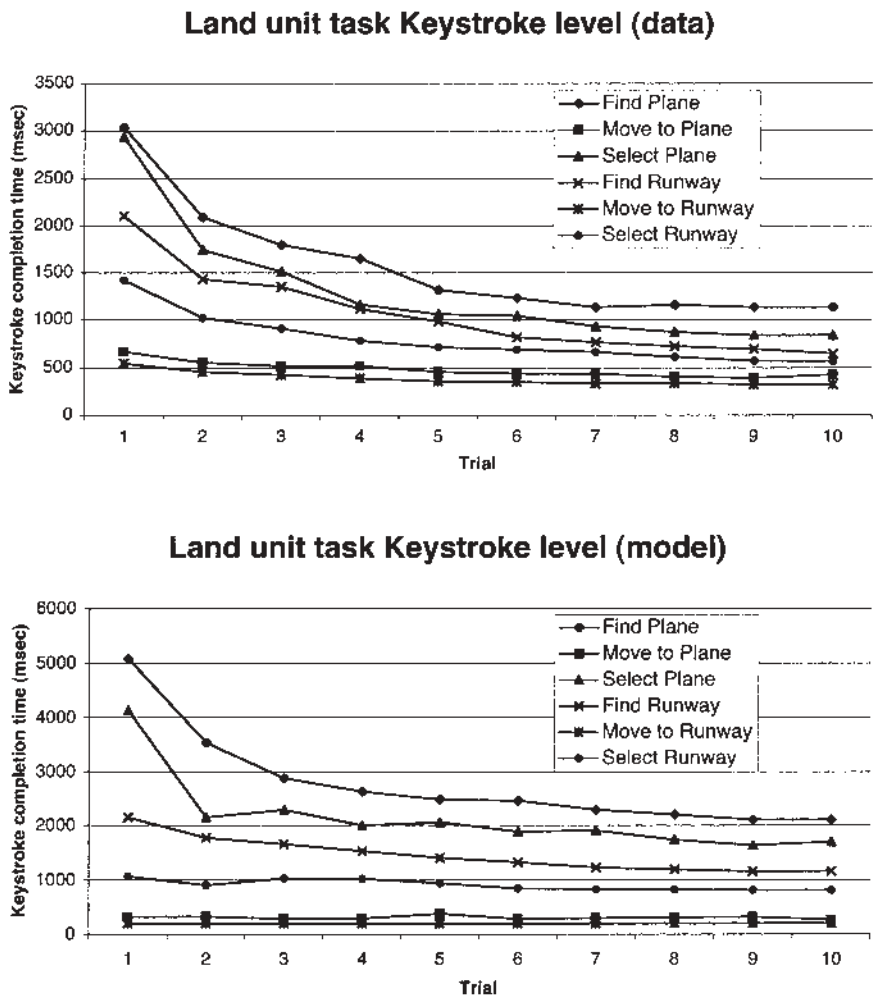


Figure 6. Keystroke-level performance on the land unit task by the participants (data, top) and the model (bottom).

Limitations of the Model

Production compilation can incorporate experiences gained during task performance, although this issue has not been explored in the present model. Taatgen (2002) showed some examples of this in an earlier ACT-R model in which illegal weather-plane-runway combinations were remembered by the model and later compiled into rules that tried to avoid these combinations.

Another aspect of learning in the KA-ATC task that is not fully explored by the present model is the tight interleaving of perceptual, motor, and declarative processes. The model has no strategies to use cognitive "slack time" (i.e., periods of time when it is waiting for one of the external actions to finish, such as moving the eyes, pushing a key, and retrieving something from memory). Such integration is necessary (Lee & Anderson, 2000) if the model is to reach the participants' level of performance after the full 18 trials.

In addition, the model starts out with too much knowledge and strategy for doing the task. Although participants presumably pick up some of this knowledge during presentation of the instructions, they still must discover and

learn much of the task knowledge, such as finding locations on the screen (Lee & Anderson, 2001), while doing the task. The fact that this strategy-discovery phase is not modeled also prevents the model from capturing the individual differences in strategy use as reported by John and Lallement (1997). Despite the lack of strategy evolution in our model, it does show one strategic transition: In the first few trials, the model uses what John and Lallement called the *sequential strategy* – that is, getting one plane from the queue, landing it, and then repeating those steps. After practice, however, this changes into the *opportunistic strategy*, in which people try to keep Hold Level 1 filled with planes but give priority to landing planes. The model's explanation is simple: At first, the model is too slow to keep the runways occupied, so when it gets a plane from the queue there is a runway available that it can use to land the plane. With practice, however, the model becomes faster and so is able to get more planes from the queue than it can actually land, thereby filling up Hold Level 1.

Despite the fact that the model currently does not capture individual differences in strategy use and tight perceptual and motor integration as a result of learning, it can be extended to incorporate them. For example, the use of slack time can potentially be modeled by incorporating production rules that act in periods of slack time (Lee & Taatgen, 2002). In addition, other strategic knowledge can be added to the model to search for certain information on the screen instead of incorporating it in the initial declarative knowledge. Finally, knowledge can be added to memorize experience and retrieve it at the appropriate moment. However, to improve on individual differences in strategy learning and use, a more in-depth investigation into how people use general procedural knowledge (e.g., analogy or other weak methods) to develop task-specific strategies is needed.

The fit of the model may also be improved by changing parameters of the model. For example, slowing down declarative retrieval may alleviate the initial discrepancy between the model and the data with respect to the number of planes landed. However, this would simply act to conceal the fact that the model is too good initially because it has more declarative knowledge than participants have.

CONCLUSIONS

In this article we have taken a task analysis for a fairly complex task, the KA-ATC task, and have combined it with the task instructions to produce a declarative representation of how this task is initially performed. This declarative representation, together with task-independent production rules and the production compilation mechanism, produced detailed predictions of learning in this task. Discrepancies between the model's predictions and the experimental outcomes can largely be attributed to our use of a single strategy for performing this task, whereas participants use a variety of different strategies and even change strategies during task execution.

Although the KA-ATC is a laboratory task, it is comparable in complexity to other tasks that have been used in human factors research (e.g., Ehret, Gray, & Kirschenbaum, 2000; Prietula, Feltovich, & Marchak, 2000). Therefore it is likely that the results of this study and the methods used in it can be generalized to other tasks and domains. The limitation of most methods of task analysis is that they describe the knowledge needed for expert behavior. As such, the main prediction about learning is limited to the view that as more knowledge is needed for a task, it takes longer to learn it all (e.g., Kieras, 1997). This prediction, however, does not take into account the fact that some knowledge may be harder to learn than other knowledge, nor does it take into account the way the knowledge is learned based on how instructions are formulated. The method used in this article gives a detailed prediction of the entire learning process, from the initial representation of the instruction to final asymptotic performance. Hence, using our approach, different types of instruction can be tested for their efficacy and different task analyses can be examined for their accuracy.

The main vehicle for our approach is production compilation. Production compilation is a powerful mechanism that combines task-specific declarative knowledge and domain-general procedural knowledge into task-specific production rules. Although it is based on earlier mechanisms from ACT* and Soar, its main advantage is that it is a single mechanism (a property it

shares with Soar) that produces small, constrained rules. It is implemented in the current ACT-R architecture. Because the model presented in this article used only default values for parameters, this led to a single prediction about behavior that we then compared with human data. This imposes a strong constraint to our model and our approach as a whole, but such constraint is useful and ultimately necessary if cognitive modeling is to be used to make a priori predictions that can be used in practice.

In order to develop the current example into a testing platform, additional work has to be done with respect to the general strategies that the model can learn and to how production compilation can be used to account for parallelizing perceptual and motor actions (see Lee & Taatgen, 2002, for some preliminary work on this). Analogy has already proved to be a strategy that works very well with production compilation (Taatgen & Anderson, 2002), and other weak methods such as means-ends analysis are probably good candidates as well (Anderson, 1987).

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.

Ackerman, P. L., & Kanfer, R. (1994). *Kanfer-Ackerman air traffic controller task© CD-ROM database, data collection program, and playback program.* Arlington, VA: Office of Naval Research, Cognitive Science Program.

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

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem situations. *Psychological Review, 94,* 192–210.

Anderson, J. R. (1990). *The adaptive character of thought.* Hillsdale, NJ: Erlbaum.

Anderson, J. R. (2000). Learning from instructions. In *Proceedings of the 7th Annual ACT-R Workshop* (n.p.). Pittsburgh, PA: Carnegie Mellon University.

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

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

Byrne, M. D., & Anderson, J. R. (1998). Perception and action. In J. R. Anderson & C. Lebiere (Eds.), *The atomic components of thought* (pp. 167–200). Mahwah, NJ: Erlbaum.

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

Ehret, B. D., Gray, W. D., & Kirschenbaum, S. S. (2000). Contending with complexity: Developing and using a scaled world in applied cognitive research. *Human Factors, 42,* 8–23.

Fitts, P. M. (1964). Perceptual-motor skill learning. In A. W. Melton (Ed.), *Categories of human learning* (pp. 243–285). New York: Academic.

John, B. E., & Lallement, Y. (1997). Strategy use while learning to perform the Kanfer-Ackerman air traffic controller task. In M. G. Shafto & P. Langley (Eds.), *Proceedings of the 19th Annual Conference of the Cognitive Science Society* (pp. 337–342). Mahwah, NJ: Erlbaum.

Kieras, D. E. (1997). A guide to GOMS model usability evaluation using NGOMSL. In M. Helander, T. K. Landauer, & P. Prabhu (Eds.), *Handbook of human-computer interaction* (2nd ed., pp. 733–766). New York: Elsevier.

Lee, F. J., & Anderson, J. R. (2000). Modeling eye-movements of skilled performance in a dynamic task. In N. A. Taatgen & J. Aasman (Eds.), *Proceedings of the 3rd International Conference on Cognitive Modeling* (pp. 194–201). Veenendaal, Netherlands: Universal.

Lee, F. J., & Anderson, J. R. (2001). Does learning a complex task have to be complex? A study in learning decomposition. *Cognitive Psychology, 42,* 267–316.

Lee, F. J., & Taatgen, N. A. (2002). Multitasking as skill acquisition. In W. D. Gray & C. D. Schunn (Eds.), *Proceedings of the 24th Annual Conference of the Cognitive Science Society* (pp. 572–577). Mahwah, NJ: Erlbaum.

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: Erlbaum.

Prietula, M. J., Feltovich, P. J., & Marchak, F. (2000). Factors influencing analysis of complex cognitive tasks: A framework and example from industrial process control. *Human Factors, 42,* 56–74.

Taatgen, N. A. (1999). *Learning without limits.* Unpublished Ph.D. thesis, University of Groningen, Netherlands.

Taatgen, N. A. (2001). Extending the past tense debate: A model of the German plural. In K. Stenning & J. Moore (Eds.), *Proceedings of the 23rd Annual Meeting of the Cognitive Science Society* (pp. 1018–1023). Mahwah, NJ: Erlbaum.

Taatgen, N. A. (2002). A model of individual differences in skill acquisition in the Kanfer-Ackerman air traffic control task. *Cognitive Systems Research, 3,* 103–112.

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

Tambe, M., Newell, A., & Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning, 5,* 299–348.

van Rijn, H., van Someren, M., & van der Maas, H. (in press). Modeling developmental transitions on the balance scale task. *Cognitive Science.*

Niels A. Taatgen received his Ph.D. in psychology in 1999 from the University of Groningen, Netherlands. He is Universitair Hoofddocent (roughly equivalent to associate professor) in the Department of Artificial Intelligence at the University of Groningen.

Frank J. Lee received a Ph.D. in cognitive psychology from Carnegie Mellon University in 2000. He is an assistant professor in the Department of Cognitive Science at Rensselaer Polytechnic Institute.

Date received: October 4, 2001
Date accepted: November 11, 2002