MODELING DUAL-TASK PERFORMANCE IMPROVEMENT:
Casting Executive Process Knowledge Acquisition as Strategy Refinement

by

Ronald Samuel Chong

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1998

Doctoral Committee:

Professor John E. Laird, Chair
Assistant Research Scientist Randolph M. Jones
Associate Professor David E. Kieras
Professor David E. Meyer

Abstract

MODELING DUAL-TASK PERFORMANCE IMPROVEMENT:
Casting Executive Process Knowledge Acquisition as Strategy Refinement

by
Ronald Samuel Chong

Chair: John E. Laird

People demonstrate a remarkable ability to perform complex, multiple-task activities in spite of the limitations of our sensory, perceptual, cognitive, and motor systems. A prominent theory that addresses how multiple-tasks activities are performed is that of the executive process. Some of the functions of the executive process include enforcing task priorities and arbitrating access to limited resources. It has been shown that a time-sharing skill (or executive-process knowledge) is acquired during training on dual-task combinations.

This dissertation presents the development of a computational, task-independent framework for modeling the acquisition of the knowledge acquired during training on dual-task combinations—executive process knowledge. On a selected dual-task combination—a continuous tracking task and a discrete two-choice reaction time task—this framework, when given the declarative and procedural representation of the novice task, has produced an expert model whose performance is a good match to empirical reaction time and tracking error data for the task combination.

There are three main contributions of this work. First is the development of EPIC-Soar, a symbolic hybrid architecture that possesses a psychologically-motivated learning mechanism and psychologically-plausible perception and motor systems. Second is the identification and classification of executive process knowledge and the taxonomies that result from this analysis. Third, is an acquisition framework which consists of: a novel data structure for representing task strategies; a task-independent procedure for resolving simultaneous access for motor resources and learning new knowledge that avoids such collisions in the future; a second task-independent learning procedure which refines the strategy data structure and creates new procedural knowledge for performing the task; and a collection of guidelines that regulate how and when promotions are applied.

To all who raised me.

Acknowledgments

In spite of how fashionable and trite it has become, I must first sincerely acknowledge God for guiding me through fifteen years of college and university—from a dismal first few years of college, to this brief moment of elation and a sense of accomplishment. He has led me to academic success, shown me the positive side of my many academic failures, opened doors of opportunity, led me to people who had my best interest at heart, and has given me insights when I have needed them most. It is for these and much more that I thank Him for this miracle He was worked in me.

Of beings corporeal, I must acknowledge my thesis committee. First and foremost to my research advisor, John Laird, who has allowed me to pursue the research issues that have motivated me, has never been hesitant to give a compliment for good work, and has always been tactful in giving reproach for poor work, or the plain lack of it. But I am most grateful for his endurance and understanding during weeks of "Sorry John, I haven't done much this week" after the birth of my daughter, Lily. It was a struggle to find a balance between fatherhood and research. Hence, of the many qualities that I hope to have adopted, the most important is John's ability to balance his devotion to his family and dedication to his research.

I'd also like to thank the rest of my thesis committee: Randy Jones for his words of encouragement along the way and his comprehensive feedback on the thesis; Dave Kieras. for many impromptu discussions about EPIC and cognitive issues in general; Dave Meyer for his stimulating and entertaining courses where he taught me everything I (should) know about cognitive psychology and human performance—any fallacies in this thesis are due to my ignorance alone and not a product of Dave's teaching. Finally, as a whole, I thank my committee for a rigorous and memorable thesis defense that highlighted the "bigger picture" that escaped me in my early draft of the thesis.

Many thanks to my parents and sister who have been unwavering in their support, encouragement, and insistence that I had what it took and would make it to the end. I'd like to also recognize my wife for the same as well as for setting aside her doctoral research to take care of Lily as I neared the end. I guess it's my turn now.

Finally, I must say a word of thanks to the many friends I've made through the SE-R (the "poor man's sports car") mailing list, and in particular, the local Michigan SE-R group—Rob R. Bob, Roger, Larry, Rob C., among many others. There were times when the research was getting the better of me. At those times, there were few diversions as enjoyable as: hanging with this group of gearheads at the Detroit Auto Show; get-togethers to change a clutch or two; resurrecting a wounded engine; upgrading a shifter in 10° weather; or pushing for a fast, clean run at the season-ending autocross at the Milford Proving Grounds. Thanks guys for helping me through this by, paradoxically, luring me away from it.

Table of Contents

List of Figures

List of Appendices

CHAPTER 1

INTRODUCTION

Humans perform many complex, high-performance multiple-task activities. Consider a few examples: a commercial pilot landing a passenger plane in inclement weather; a military pilot engaging or evading a foe. A personal favorite is: a Formula 1 race car driver approaching a hairpin corner and needing to: downshift to an appropriate gear for the exit of the corner; threshold brake to the maximum speed for the corner; select the appropriate racing line for the corner taking into consideration the next corner; all while planning race strategy on the radio with the pit crew. Two less glamorous examples are: piano players repeating prose that is read to them while they are sight reading materials of various difficulty (Allport, Antonis & Reynolds, 1972), and subjects reading text aloud while taking dictation (Spelke, et al., 1976).

Some of these activities share several points in common. First and foremost, the individuals performing these activities have before them a number of tasks that must be performed simultaneously. Second, some of these activities may have a strong perceptual-motor flavor; i.e., the perception (visual perception, for example) of external events is used by cognition to elicit overt behavior in the form of motor actions, such as eye movements, button presses, or steering corrections. Because of this, strictly simultaneous performance is not always possible since, when generating overt behavior, cognition must take care not to violate the structural limitations of the effectors. For instance, we cannot fixate our eyes in more than one direction at a time nor can a hand be made to move in two directions at the same time.

A third commonality among some of these tasks is that there are intrinsic time pressures associated with each task. One characteristic of high-performance tasks is that behaviors in response to stimuli must flow as quickly as possible. A fourth and final commonality is that there may be a potentially high cost for failure to perform or just in *delaying* to perform the task.

These points together illustrate why it can be very challenging to perform multiple-tasks at an expert level. Yet, as the previous examples have shown, people can perform amazingly well under multiple-task conditions. How is this done? How do people manage the different requirements for performing a multiple-task combination? More fundamentally, given that all experts were once novices, how do people learn to perform these combinations? Additionally, what do they learn through practice on a combination that enables them to become experts?

1.1 RESEARCH OBJECTIVE

To date, no work has been done toward modeling the acquisition of knowledge through practice on a dual-task combination. The work presented in this thesis begins to fill this void. Therefore, our research objective is to:

Develop a computational, task-independent framework for modeling the acquisition of the knowledge that subjects acquire through practice on a selected dual-task combination.

Ideally, one would want to evaluate such a framework on at least these two dimensions: its ability to a) produce models that transition from novice to expert performance, *and* b) predict the time to make the transition; the learning time. This work however is only concerned with the former and not the latter. The reason is two-fold. First, we have been unable to find simple perceptual-motor task combinations for which trial-by-trial learning data was collected. Second, it was anticipated that the first of these attributes would be a significant achievement on it own without the added constraint of the latter.

We will also evaluate this framework on its ability to construct models that produce a quantitative post-learning match to the observed expert performance data for the selected task, given plausible initial knowledge. We will also evaluate it based on its psychological plausibility and the predictions it makes.

This work focuses only on simple perceptual-motor tasks in contrast to cognitive tasks such as list memorization, memory search or counting where there is no overt behavior. Specifically, we will concentrate on visual-manual tasks—tasks that take in visual stimuli (such as the onset and identity of objects) and produce manual responses (such as keypresses or joystick moves)—because: a) these are the most prevalent kinds of tasks, and b) in the architecture that will be used for modeling, the visual and manual modalities are the most developed in comparison to the auditory and vocal modalities.

1.2 Research approach

The approach used to address the research objective consists of two steps. First, we aim to identify the executive process knowledge that is sufficient to produce expert performance on a selected task. To accomplish this, we start with a novice model of the individual tasks, then iteratively elaborate the models, using a generate-and-test methodology, until a model is found whose performance matches the observed expert performance data. When elaborating the models, we are careful to adhere to a fundamental tenet of cognitive modeling: that the modeler should be able to articulate the source of *all* the knowledge in the model. Once the executive process knowledge has been identified, we hope to identify a few broad categories for the knowledge.

The next step in the research approach is to devise some learning procedures or possibly a general framework that supports the acquisition of the identified classes of knowledge. Though this approach seems analogous to data-fitting (producing learning procedures to learn the desired knowledge), it is hoped that the knowledge classes and the learning procedures or the framework will be sufficiently general to apply to a large class of tasks, while making confirmable and/or plausible predictions about human dual-task acquisition and performance.

1.3 Why study dual-task acquisition?

Of the many reasons to study human dual-task acquisition, the first is the most obvious: like most of human behavior, it remains an intellectual challenge to discover how humans do it. There are many interesting questions that are unanswered. What training regimen will produce the best final dual-task performance? What are the aspects of a task combination that prevent certain combinations from being performed as well as others? Similarly, what are the affordances that allow other combinations to be performed well together? Why are some people (e.g. fighter pilots) significantly better at performing under dual-task conditions than are others (e.g. those who flunk fighter pilot school)?

Many of the high-performance activities mentioned at the beginning of this chapter can produce high levels of mental workload (Gopher & Donchin, 1986). When the workload reaches very high levels, an individual may resort to undesirable ways of coping. A second reason to study dual-task performance is that as the details of human dual-task acquisition and performance are revealed, methods may be devised for reducing the sometimes significant mental workload subjects experience when performing multiple tasks.

Consider the situation faced by military pilots—tones from onboard missile systems; tones warning of enemy missile locks, radio communications from a wingman or backseat navigator, avoidance of AAA (anti-aircraft artillery) fire and SAMs (surface-to-air missiles), not to mention the necessary task of keeping the aircraft in the air and navigating to the target in a timely fashion. There have been reports of Vietnam War fighter pilots who, while in the throes of the battle and faced with complete sensory and task overload, would shut off warning systems so that they could concentrate on whatever task was considered to be of foremost importance. As the details of dual-task acquisition and performance are revealed, they may lead to better designed systems or equipment, or may result in training regimens and performance techniques to reduce the workload on the individual performing the task.

A third reason to study dual-task acquisition is economic concerns. A clearer understanding of how people learn dual tasks can imply a significant financial savings. One could develop more streamlined and effective training and testing techniques. Also, with a clearer understanding of how people then go on to perform dual tasks, more savings can be realized in the design and engineering of systems that people must operate. The often cited example is the analyses of the telephone operator task (Gray, John & Atwood, 1992). The analyses resulted in a small reduction in average task completion time. However, the domain is such that small time savings translate into substantial monetary savings.

A final reason is that the dual-task situation can give insights into the architecture of the mind. These types of situations stress human capabilities very seriously, and so the observed patterns of behaviors set very strong constraints on the human information-processing system architecture. Therefore, analyses of even simple dual-task situations can result in detailed hypotheses about information processing mechanics (Kieras & Meyer, 1995a).

1.4 Thesis summary

This section summarizes the main body of work presented later in the thesis. Each subsection that follows is a synopsis of the chapter with the same name. This summary can be read as an introduction to, or in lieu of, the remainder of the thesis.

This summary and the thesis as a whole are arranged in a chronological fashion, reflecting the order in which the original research was performed. This is an appropriate organization as it follows the logical development of the ideas presented.

1.4.1 The psychology of multiple-task behavior

How are people able to perform multiple, simultaneous tasks? The most prominent theory of multiple-task performance posits that performance and resource management are accomplished by an *executive routine*, or *executive process* (Borkowski and Burke, 1996; Gopher, 1993; Meyer & Kieras, 1997a; Neisser, 1967; Norman & Shallice, 1986).

Abstractly, executive process knowledge might be defined as the knowledge necessary to produce dual-task performance that is distinct from the knowledge used to perform the constituent tasks. Meyer and Kieras (1997a) provide a concrete description of the role of the executive process. It maintains task priorities and coordinates progress on concurrent tasks through various types of supervisory control. It possesses no knowledge for performing any of the individual tasks, but it does have knowledge about the relative and/or absolute priorities of the tasks under its control. It

must also arbitrate when two tasks cause a violation of the structural limitations of the effectors; e.g., `TaskA` wants the eye to look to the left, while `TaskB` wants the eye to look to the right. The role of the executive process is in many ways similar to that of computer operating systems such as UNIX.

Where does this executive process come from? Damos & Wickens (1980) developed a measurement technique to demonstrate clearly the acquisition of a *timesharing skill* (or executive process knowledge) under dual-task training. With this technique, they showed convincingly that there are distinct timesharing skills developed during dual-task training.

Empirical studies have consistently shown that when even the simplest tasks are performed concurrently, there is *usually* a reduction in performance of both tasks compared to when each task is performed alone. This worsening in performance has been called the *dual-task decrement* and is assumed to be due to *dual-task interference*.

Many theories have been proposed in an attempt to explain the cause of this interference. (See Meyer & Kieras (1997a) for a thorough discussion). Many theories have assumed that there is a limitation in the processing chain from stimulus perception through response generation (Broadbent, 1982; Keele, 1973; Pashler, 1984, 1990, 1992, 1994a, 1994b; Welford, 1952). Others have posited the existence of a limited resource/s that must be shared by the competing tasks (Gopher & Donchin, 1986; Kahneman, 1973; Wickens, 1980, 1987, 1991). Recently, Meyer & Kieras (1997a) have suggested the "radical" idea that there is no inherent cognitive bottleneck. At times, one of these hypotheses has seemed to prevail, whereas other evidence has favored other. The debate continues.

1.4.2 Selecting an appropriate computational architecture

The goal of this work is to develop a computational, task-independent framework for modeling the acquisition of executive process knowledge. As such, a computational architecture for the exploration and development of this framework is needed.

Based on the subgoal of the overall research goal, we defined requirements on the ideal architecture's knowledge, cognitive, and performance systems. The knowledge system should support the declarative vs. procedural distinction along with representing this knowledge in an inspectable way; i.e. symbolic representations. The cognitive system should provide a learning capability along with the absence of an inherent cognitive bottleneck. The performance system should provide a psychologically-plausible account of perceptual and motor abilities and limitations.

We discovered however that no single architecture provided complete coverage of all these requirements. However, EPIC (Meyer & Kieras, 1997a, 1997b) and Soar (Laird, Newell, & Rosenbloom, 1987) architectures together address all the requirements. EPIC possesses the most thorough computational theory of perceptual and motor processors, yet EPIC's cognitive processor does not learn. Soar, on the other hand, represents one of the most complete computational theories of the human cognition and has not inherent cognitive bottleneck. It also has a single mechanism for learning, but does not have a theory of sensory or motor processes that is as mature as those for its cognitive processor.

The potential for a synergistic merging of EPIC with Soar was obvious and the endeavor was undertaken. The resulting hybrid architecture is called EPIC-Soar (Chong, 1995; Chong & Laird, 1997; Chong, 1998a, 1998b). EPIC-Soar represents a parsimonious integration of the perceptual and motor processors of EPIC with Soar. This merger is an attempt to get both the detailed predictions and explanations provided by the perceptual and motor processors of EPIC (an ability Soar does not possess) and the problem solving, planning, and learning capabilities of Soar (an ability EPIC does not possess).

1.4.3 The Wickens task

A task combination we call the *Wickens task* (Martin-Emerson & Wickens, 1992) was chosen because an expert model of this task already existed in EPIC. It consists of a continuous tracking task and a choice-reaction time task. This task combination was originally used to evaluate the effect of vertical separation (between the tracking and choice tasks) on tracking performance. The application of this study is to the design of heads-up displays used in military and civil aviation.

1.4.4 EPIC on the Wickens task

EPIC has been used to produce a quantitatively accurate model for the Wickens task (Kieras, 1994; Kieras & Meyer, 1995a). The production rules of the EPIC model realize a model of *expert* dual-task performance for the task. The EPIC model provides a good overall match to the empirical data on both the reaction-time and tracking error measures. This success is a direct result of combining psychologically realistic perception and action with cognition.

Concerning the executive process, some of the rules were task facilitation rules that moved the eye between the tasks. Other rules however, are best defined as implementing an explicit, task-specific strategy for coordinating the two tasks. Tasks are disabled and enabled for fundamental reasons (such as preventing motor conflicts) and for task performance reasons (such as disabling the tracking task because the eye was away from the cursor). In a sense, these rules "micro-manage" the execution of the tasks, providing deliberate control of how the steps of each task are interleaved and as such, they are task specific.

1.4.5 The classification of executive process knowledge

This chapter presents the first of three main phases of this work. This first phase had four objectives. First, to construct a Wickens task model in EPIC-Soar since all the future work would be done in EPIC-Soar. Second, to confirm that the EPIC-Soar hybrid architecture was sufficient for modeling dual-task expert performance. Third, to explore the possibility of using a task-independent, minimalist executive process instead of EPIC's task-specific formulation. The fourth objective was a pragmatic one: building a performance model would allow us to: a) identify the knowledge that needed to be learned, and b) reverse engineer learning procedures for acquiring the identified knowledge.

To create the expert dual-task performance model in EPIC-Soar, we first started with expert models of the individual tasks. To build the first dual-task model, we loaded both individual task models into EPIC-Soar and added some knowledge to the system to implement the basic lockout strategy. We ran this model and collected the data. The performance was found to be rather poor. A little more knowledge was added to the model and it was re-run. This generate-and-test cycle was repeated until an expert model was found.

This incremental approach yielded a lineage of four dual-task models, ending with a final expert dual-task model. The changes/additions that were made at each increment represent the executive process knowledge used for this task. This knowledge was partitioned into two sets indicating when the knowledge was acquired; either before or after the first dual-task trial was performed. Knowledge that existed before the first trial was called *pre-trial*. Knowledge that resulted from performance on the task was called *learned*. The knowledge in the latter class is what should be learned by the sought-after learning procedure/framework. The learned knowledge class consisted of two general types of rules: anticipatory motor programming rules and pipelining rules.

1.4.6 A framework for acquiring executive process knowledge

In this chapter, we present the development of a framework for acquiring these rules. First, a key observation about these rules is presented, followed by our hypothesis about the initial knowledge

that subjects possess. The observation and hypothesis together suggested a task-independent learning procedure.

When comparing the novice dual-task rules with the expert rules, a key observation was made: the expert rules produce improved performance by executing commands, or parts of commands, *chronologically earlier in the task*, as though they were *chronologically promoted*. In addition to anticipatory motor programming and command pipelining, a third task independent method for improving performance is called *movement pre-positioning* (Wood, Kieras, & Meyer, 1994; Kieras, Wood & Meyer, 1995a, 1995b). The idea here is that performance improvements can be had by positioning an effector (a hand or an eye, for example) at a location before the action at that location is needed. This movement is clearly a kind of promotion.

We next asked and answered a fundamental question: "What initial knowledge does a subject possess *after* receiving task instructions but *before* beginning to perform a task?" We made two assumptions about subjects: a) that they understand the task instructions, and b) are sufficiently motivated to perform the task and abide by the instructions. Our answer to this question was that this knowledge is a detailed plan, or strategy, of how to perform the task. This strategy was realized by a data structure called a *chronological task strategy data structure*. This structure represents a subject's knowledge about the chronological ordering of perceptual events, and the motor commands that are required (per the task instructions) at the occurrence of each perceptual event. The structure also captures a subject's interpretation biases, and pre-performance reasoning about the task.

With a structure such as this, *principled* chronological promotions could be performed. A straightforward task-independent *promotion-learning* procedure for acquiring the learned knowledge was devised. The procedure performs three styles of promotions: *prepare promotions*, *event promotions*, and *chain promotions*. Prepare promotions create anticipatory motor programming rules, event promotions create movement pre-positioning rules, and chain promotions create pipelining rules.

The resulting framework consist of: a novel data structure for representing task strategies; a task-independent procedure for resolving simultaneous access for motor resources and learning new knowledge that avoids such collisions in the future; and a second task-independent learning procedure which refines the task strategy data structure and creates new procedural knowledge for performing the task, and a collection of guidelines that regulate how and when promotions are applied.

To apply the framework to a task, the modeler must provide two things to the framework: a) a chronological task strategy data structure, hand-coded as declarative knowledge, and b) a procedural performance model that implements the behavior represented in the strategy structure. This information was provided to the framework for a fictitious choice reaction time task. The learned expert model was shown to produce plausible post-learning performance.

1.4.7 Applying the acquisition framework to the Wickens task

In this chapter, the acquisition framework is applied to Wickens task. A chronological task strategy data structure for the task was hand-coded as declarative knowledge and given to EPIC-Soar. In addition, a novice dual-task model, whose behavior was congruent with the strategy structure, was given to EPIC-Soar. After training, the learned expert model's performance was a good match to the human data.

1.4.8 Predictions of the acquisition framework

The models in earlier chapters assumed that novice subjects initially use a lockout dual-task strategy then eventually progress to an interleaved dual-task strategy. Realizing the some subjects may never actually use a lockout strategy but rather might begin with an interleaved dual-task strategy,

we decided to re-run the model using this configuration. Our expectation was that the promotion learning procedure would produce *exactly* the same end performance, making the prediction that the dual-task strategy initially used did not matter to final performance.

To confirm our expectation, the EPIC-Soar system was run in this configuration and trained. Contrary to our expectations, we found that the final performance was in fact *not* the same; specifically, the tracking error performance was worse. The framework and model made the prediction that: a) post-training performance is dependent on the dual-task strategy used during training, and b) an initial-lockout strategy leads to better performance than an initial-interleaving strategy.

We propose that this prediction is supported by the results reported by Gopher (1993) about a study that examined the effect of varied task emphasis in dual-task learning. It was found that post-training performance was superior for the subject group that varied priorities during learning (the VP group) when compared to subjects who were either in the equal-priority (EP) or no-priority (NP) groups. The initial-lockout model (which initially emphasized the choice task, then later changed to equal-emphasis when switched to use an interleave strategy) is superficially like the VP group which gave each task varying levels of emphasis. The initial-interleaving model is superficially like the EP group in that both used equal-emphasis throughout. Therefore, it appears that the model has made a confirmed prediction.

Gopher (1993) did not report or speculate on how the relatively poor performance of the equal-priority and no-priority groups could be elevated to reach the superior level of the varied-priority group. However, our acquisition framework suggests a simple retraining regimen that would enable the initial-interleaving model (which is analogous to the EP group) to reach the performance of the initial-lockout model (which is analogous to the VP group). It is expected that performance would improve if the model were given some practice trials in the VP condition. This prediction was tested and the retrained performance was comparable to the final results for the initial-lockout model.

1.4.9 Discussion

This chapter discussion of various aspects of the acquisition framework. Theoretical issues about this work and the architectures involved are also discussed. Finally, the implications of this work are presented.

1.4.10 Contributions

This work makes four contributions. First is the development of the EPIC-Soar hybrid architecture for learning and performance. The second is the executive process knowledge taxonomies that was created based on the knowledge classes and subclasses identified in the work. Third is the acquisition framework, which consists of: the jam-recovery learning procedure and the jam-avoidance procedure; the chronological task strategy data structure; the promotion learning procedure; and the promotion procedure's application guidelines. The final contribution is that this framework may automate the search for the fastest-possible strategy model used in the bracketing heuristic devised by Kieras & Meyer (1998).

1.4.11 Future work

Further validation is needed for the task models used in this thesis. For the example CRT task of Chapter 6, empirical data needs to be collected to confirm that the pre- and post learning performance of the model is reasonable. For the Wickens task, the individual task models need to be validated against empirical performance data of the individual tasks.

Further validation of the acquisition framework is also needed. This work focused only on the transition from novice to expert, not on the *time* to make the transition. Empirical learning studies

should be done to compare the learning times (possibly in terms of trials). If the predicted times are not acceptable, then extensions to this framework may be needed.

Many researchers have shown that people use different task strategies while performing and while learning to perform tasks. One of the contributions of this work is the proposed structure for representing task strategies. It may be possible for the acquisition framework to be used as a foundation for a system to exploring the question of how people generate and select task strategies.

Before the framework can be applied to a task, the modeler must provide the chronological task strategy data structure for the task, and procedural knowledge that performs the task exactly as it is represented in the structure. An area for future work may be to work toward removing these requirements. Instead, of providing the procedural knowledge, one could develop a new learning procedure that would learn the procedural knowledge from the declarative strategy structure. There has been much work in this area of the acquisition of skill and knowledge compilation. (Anderson, 1983, 1987; Anderson et al., 1981; Neves & Anderson, 1981). There is also prior Soar work by Huffman (1994) that touches on this topic.

Additionally, one could go a step further and develop a learning procedure that would accept plain text instructions as input and convert them into a declarative representation; specifically, our strategy data structure. This development would obviate the need for the modeler to provide the strategy data structure. There is previous work in Soar on instructable autonomous agents (Huffman, 1994) and on natural language comprehension (Lewis, 1993). These successful works may be recruited to this effort. There is also non-Soar work in the area of instruction following by Bovair & Kieras (1991) that speaks to this specific issue.

By finding and incorporating such learning procedures into the framework, EPIC-Soar would be able to model a much broader novice-to-expert transition. It would also provide further confirmation that Soar's chunking mechanism is sufficient for modeling all the learning along a common trajectory of human experience.

CHAPTER 2

The psychology of dual-task behavior

2.1 The executive process

How are people able to perform multiple, simultaneous tasks? How do they manage the potentially conflicting demands on limited cognitive and physical resources? The most prominent theory of multiple-task performance posits that performance and resource management are accomplished by an *executive routine*, or *executive process* (Borkowski and Burke, 1996; Gopher, 1993; Meyer & Kieras, 1997a; Neisser, 1967; Norman & Shallice, 1986). Eslinger (1996) says "executive functions are considered by many scientists to be one of the crowning achievements of human development."

Abstractly, executive process knowledge might be defined as the knowledge necessary to produce dual-task performance that is distinct from the knowledge used to perform the constituent tasks. Figure 2.1 illustrates this idea. Initially, a subject has distinct knowledge sets for performing `TaskA` and `TaskB`. To get dual-task performance with both tasks generally requires some *additional* knowledge; this is the executive process knowledge. Together, these three knowledge sets allow skilled dual-task performance.

Meyer and Kieras (1997a) provide a concrete description of the role of the executive process. It maintains task priorities and coordinates progress on concurrent tasks through various types of supervisory control. It possesses no knowledge for performing any of the individual tasks, but it does have knowledge about the relative and/or absolute priorities of the tasks under its control. It must also arbitrate when two tasks cause a violation of the structural limitations of the effectors; e.g., `TaskA` wants the eye to look to the left, while `TaskB` wants the eye to look to the right. The

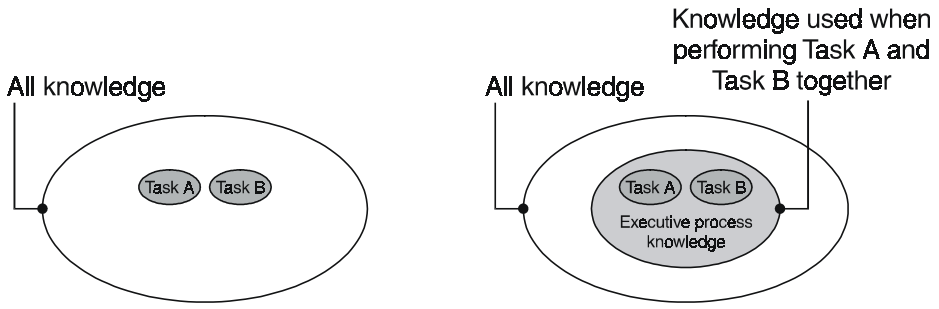


Figure 2.1 A representation of executive process knowledge

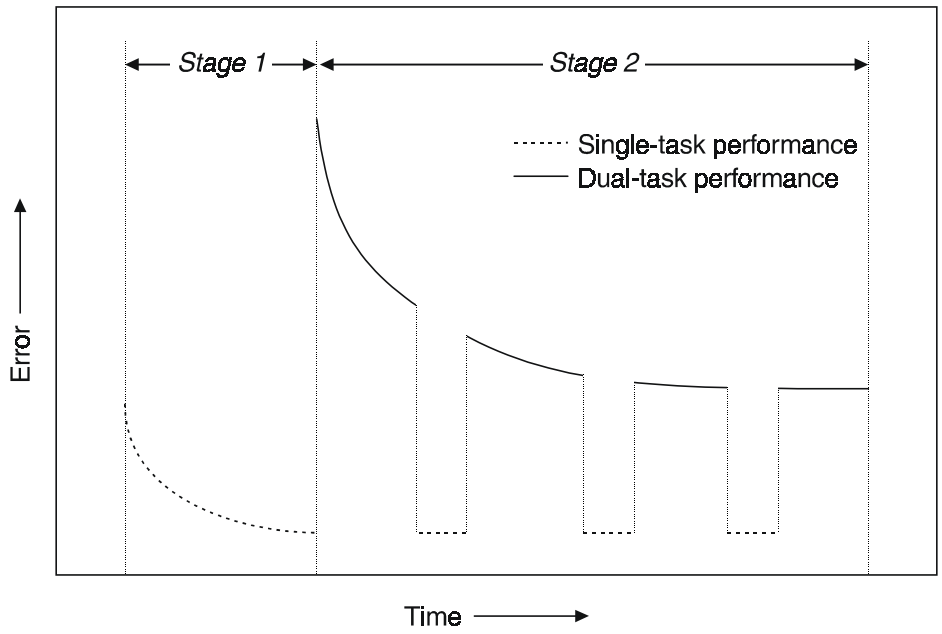


Figure 2.2 A measurement technique for identifying the acquisition of a time-sharing skill

role of the executive process is in many ways similar to that of computer operating systems such as UNIX.[1]

2.2 Acquiring a time-sharing skill

Where does this executive process (or more generally, the knowledge used by the executive process for a specific task) come from? Anyone who is an expert at a complex, multiple-task activity would recount that when they first performed the task, it was very difficult. After much practice (which may range from minutes, hours, or even years) they were able to perform the task expertly. It seems clear that executive process knowledge is the product of extensive practice in the dual-task situation.

Damos & Wickens (1980) address this question of learning an executive process. They developed a measurement technique to demonstrate clearly the acquisition of a *timesharing skill* under dual-task training. A timesharing skill is the skill one develops and uses to perform proficiently under multiple-tasks conditions. Hence, it is synonymous with executive process knowledge.

Figure 2.2 depicts their measurement technique. It shows the performance levels for only one task, say `task1`, under two performance conditions. Stage 1 represents the single-task condition where `task1` is performed alone until performance reaches some stable level. Once it has stabilized, the dual-task performance condition, Stage 2, begins. In this stage, the subject must simultaneously perform `task1` with a second task.

There are three noteworthy aspects of Stage 2. First, performance of `task1` at the beginning of Stage 2 is markedly decreased compared to the end of Stage 1; the single-task performance condition. Next, performance gradually improves with practice in the dual-task condition. The final aspect is perhaps the most important. At certain times during dual-task practice, single-task performance of `task1` is re-accessed to confirm its stability.

Taken together, this figure suggests that if the performance under the single-task condition during Stage 2 produces the same level of performance as at the end of Stage 1, then the improvement in performance noted during Stage 2 may be attributable solely to the development of a timesharing skill. In other words, since the performance of the individual tasks remain the same, the changes may be due to the subject learning how to better perform the two tasks together.

2.3 Executive process or new skill?

Damos and Wickens (1980) showed convincingly that during dual-task training, while there is speedup within the individual tasks, there are distinct timesharing skills that are also being developed. A follow-up question then is: what is the nature of this timesharing skill? There may be two possibilities. The timesharing skill could be a new skill that is a composition of the individual tasks; a new integrated skill. Alternatively, the timesharing skill could be a task-independent executive or supervisory process that mediates between the two tasks.

Damos (1991) relates an experience from a study in dual-task behavior. She developed a dual choice-reaction time experiment where subjects responded to stimuli '1' thru '4' with the left hand and '5' thru '9' with the right hand. Both tasks were unpaced (the subjects were not forced to response within a time limit but rather was allowed to produce a response at their leisure). Also, the subjects received little single-task practice before beginning the combination. Although the subjects received no instructions about integrating the information content of the two tasks, they rapidly attained a level of performance that matched single-task levels.

The data showed that the subjects were responding to the stimuli simultaneously, causing the initial interpretation to be that subjects had learned to timeshare the tasks almost perfectly. It was quickly pointed out that the subjects were probably no longer performing two independent tasks, but rather had combined the two tasks into one, more complex task. According to Damos, further experimentation showed that this was the case. Damos' experience demonstrates that under certain dual-task conditions—such as when the stimuli for the two independent tasks occur simultaneously—subjects appear to fold independent tasks into a single, more complex new skill.

Damos' results could also be couched in terms of an executive process rather than in terms of the acquisition of a new skill. For instance, subject may have learned an executive process that allowed them to perform these tasks efficiently and simultaneously. This description is consistent with the role of the executive process.

It seems reasonable that the human acquisition system could generate both forms of skill. An integrated skill could be developed when the task environment and task instructions promote integration. For example, if stimuli are presented simultaneously, the subject could compose the separate stimuli into one. When this isn't possible, or when the nature of the task is such that priorities fluctuate, then an executive process may be learned.

2.4 Dual-task interference

Empirical studies have consistently shown that when even the simplest tasks are performed concurrently, there is *usually* a reduction in performance of both tasks compared to when each task is performed alone.

An example of this decrement in performance is the large difference in stable performance as seen in Figure 2.2. (Though this graph does not depict actual performance results for a task, it is representative of the behavior seen in empirical studies.) Recall that this graph is representative of the performance of one task, `task1`, under single and dual-task conditions. This worsening in performance has been called the *dual-task decrement* and is assumed to be due to *dual-task interference.*

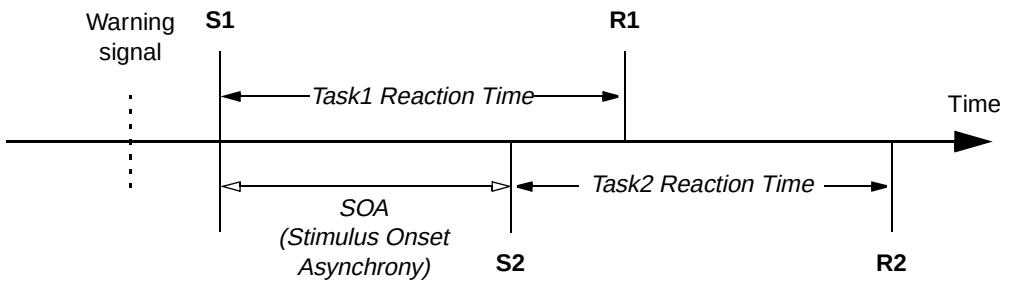


Figure 2.3 Outline of the events that occur on each trial of the PRP procedure

Many theories have been proposed in an attempt to explain the cause of this interference. (See Meyer & Kieras (1997a) for a thorough discussion). The earliest of these theories was the *global single-channel hypothesis*, a term coined by Meyer & Kieras. This hypothesis states that all of the mechanisms between stimulus input and response output (stimulus perception, response selection, movement initiation) together constitute a single-channel and can be used by only one task at a time (Craik, 1948; Welford, 1952). Therefore, in this theory, dual-task interference results from the postponement of a task from entering the channel due to another task already being in the channel.

Some tests of this and other interference theories came from a now commonly used laboratory experiment called the *psychological refractory period*, or PRP procedure. In a typical PRP experiment, two stimuli, S1 and S2, are presented and are separated by a varying period of time known as *stimulus onset asynchrony* or *SOA*. The subject makes a response to each stimulus but is instructed that the response to S1 must precede the response to S2. See Figure 2.3.

A plot of the reaction time results (Figure 2.4) illustrates that the task1 reaction time is independent of the SOA as one would expect. However, note that as the SOA increases, task2 reaction time decreases. This is the so-called *PRP effect*.

Later research with the PRP procedure cast doubt on the global single-channel hypothesis. It was found that the PRP effect at zero SOA did not always equal mean task1 RTs and sometimes is significantly less than this hypothesis could predict. This suggests that the single channel does not involve all intervening processes. Theorists have therefore looked for some specific stage of processing that could constrain multiple-task performance.

2.4.1 Perceptual-bottleneck model

Under the perceptual-bottleneck model, the process that identifies stimuli and assigns semantic meaning to them is limited. For concurrent tasks, this limit would force people to deal with only one task at a time. This model makes no specific claims about what, if any, constraints exist on subsequent processes after stimulus identification, so it has been called the *early-selection theory*.

One prominent exemplar was Broadbent's filter theory (Broadbent, 1982). He proposed that sensory stimuli first enter a sensory buffer where they undergo initial feature analysis during which physical features are analyzed and made available to a selective attention filter. Broadbent supported his assumptions by citing experiments in choice RT, dichotic listening, and oral shadowing.

Soon afterwards, other studies revealed significant counterevidence. Treisman (1960, 1969) showed that under some conditions, subjects notice significant amounts of semantic information in unattended auditory messages. This is an example of the classic "cocktail party" effect: even while engrossed in an interesting conversation, one's attention will be captured by the unexpected

mention of one's own name (or an expletive) in a distant conversation. This demonstrates that semantic process is going on for both attended and unattended stimuli. There cannot be an early selection bottleneck. Consequently, some theorists have proceeded to look beyond the perceptual processes for bottlenecks elsewhere in the human information processing system. These post-perception theories can collectively be called *late-selection theories*.

2.4.2 Response-selection bottleneck hypothesis

The most prominent late selection theory is the response-selection bottleneck (RSB) hypothesis. It is based on the idea that parallel processing may be possible for some mental operations but is impossible for others, thus causing a bottleneck. Hence, the RSB hypothesis states that the limits seen in human performance in the PRP procedure are due to a bottleneck in the response-selection stage.

This explanation originated with Welford's investigations (Welford, 1952). For a time, studies into this phenomenon ceased, but they have since been revisited and continued by Pashler (1984, 1990, 1992, 1994a, 1994b) and others.

The RSB hypothesis predicts many salient properties of the idealized PRP curve. First is that mean `task1` RTs are affected by neither the SOA nor `task2` difficulty. Since there is a response-selection bottleneck, as long as the stimuli for `task1` appears first, its mean RT should stay constant. Second, the mean `task2` RTs are higher at short SOAs. Since a `task2` response cannot be selected until `task1` has made its response selection, `task2` RT will certainly be longer than `task1` RTs. Third, as SOAs gradually increase, mean `task2` RTs decreases. Under the RSB hypothesis, the slopes of these curves should equal -1 for short SOAs because each increment in SOA (starting from zero) produces an equal opposite decrement in how long a `task2` stimulus must wait to enter the response-selection bottleneck. A final property implied by the RSB hypoth-

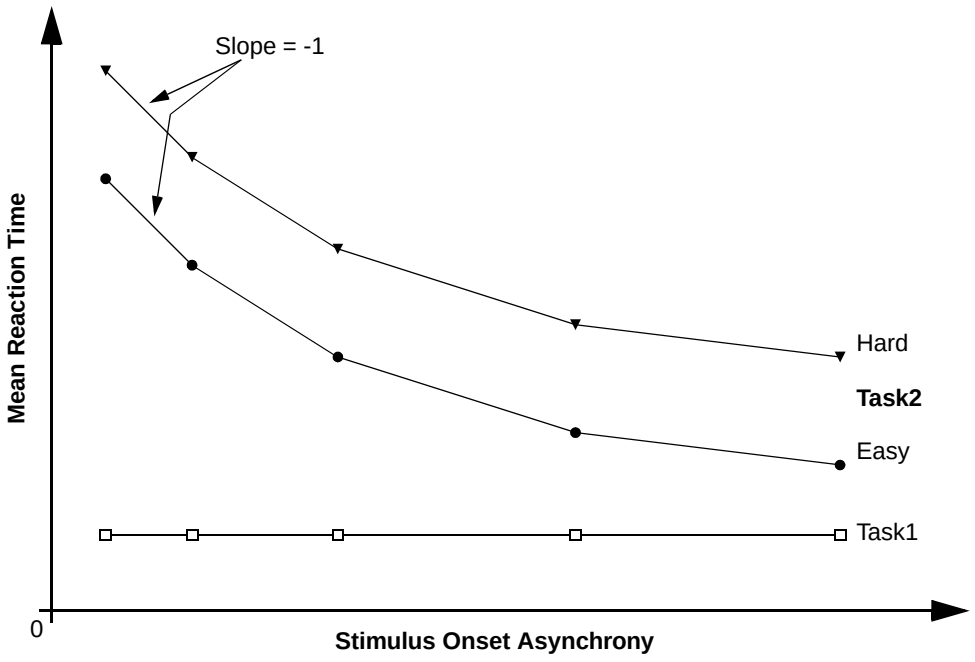


Figure 2.4 Idealized representative results from the PRP procedure (Meyer *et al.*, 1995)

esis is that the SOA and response-selection difficulty of `task2` should affect mean `task2` RTs additively. As a result of this, when the difficulty of `task2` response selection is manipulated, parallel `task2` PRP curves should be observed, as seen in Figure 2.4.

While the RSB hypothesis can account for the typical PRP effect, there are many studies that have manipulated PRP variables and have produced performance curve characteristics that are inexplicable within the hypothesis. We will discussion three of these characteristics in turn: divergent PRP curves, convergent PRP curves, and initial slopes steeper than -1 (Meyer *et al.*, 1995).

Hawkins, Rodriquez & Reicher (1979) manipulated `task2` response selection by manipulating the number of S-R pairs in `task2`. In this study, it was found that the `task2` difficulty effect is only about 25ms at the shortest SOAs, but grows to nearly 200ms at the longest SOA.

Irvy *et al.* (1994) manipulated response selection by varying the spatial S-R compatibility in `task2`. At the shortest SOA, the difficulty effect on mean `task2` RT is nearly 300ms, whereas at the longest SOA, it is less than 200ms. These two studies demonstrate positive and negative SOA-difficulty interaction, respectively. Together, they clearly raise some doubt about the validity of the RSB hypothesis since it claims that `task2` difficulty should be strictly additive (no SOA-by-difficulty interaction), producing parallel PRP curves across all SOAs.

Finally, Lauber *et al.* (1994) manipulated `task2` similar to Hawkins *et al.* (1979), but they also manipulated the number of `task1` S-R pairs. This change yielded "parallel" average PRP curves with approximately additive effect of SOA and task2 difficulty. However, over the two shortest SOAs, Lauber *et al.* (1994) found that the PRP curves had extremely negative slopes (almost -1.4) and were reliably steeper than -1. This result, like the others, cannot be explained by the RSB hypothesis.

2.4.3 Movement-initiation bottleneck hypothesis

A less prominent late-selection theory is the movement-initiation bottleneck hypothesis (Keele, 1973). According to this hypothesis, both perception and response selection can progress in parallel and be completed without mutual interference to concurrent tasks. However, there is assumed to be a subsequent process that initiates movements separately for each task and that can deal with only one task at a time. This would create a bottleneck wherein movement initiation for a higher-priority first task proceeds from start to finish while the initiation of another movement for a lower-priority second task waits temporally until the first is completed.

2.4.4 Unitary-resource theory

There are other theories that attempt to explain dual-task interference without making assumptions about bottlenecks at different stages of processing. One such theory is the unitary-resource model (Gopher & Donchin, 1986; Kahneman, 1973). According to this hypothesis, the central processes involved in response selection, movement initiation, and so forth are not necessarily restricted to dealing with only one task at a time. Instead, it is assumed that people have a single, finite reservoir of processing capacity, which may be allocated flexibly and equitably among various activities, depending on prevailing level of psychological arousal and task demands.

2.4.5 Multiple-resource theory

A second theory, an extension of the unitary-resource model, posits that various disjoint complementary sets of processing resources are assumed to be used in performing individual tasks. Wickens (1980, 1987, 1991) proposed three plausible candidates for the structural composition of resource reservoirs: stages of processing, cerebral hemispheres, and modalities of processing (both encoding and response).

The classification by stages or processing matches the architectures of the processing systems as it emerges from performance experiments. Hemispheres of processing are suggested from theo-

retical analysis and experimental work that views the cerebral hemispheres as acting partially as separate resource reservoirs by virtue of their functional and spatial separation. Finally, modalities of processing are justified by studies that compare auditory and visual modes of presentation and verbal versus manual modes of response in dual-task paradigms.

In this theory, each set of resources has its own separate, divisible source of capacity. If two tasks require the same set of resources, then the capacity available to them may be allocated in a flexible graded fashion, depending on current task requirements. In contrast, if two tasks require entirely different sets of responses, then progress on them may proceed simultaneously without any interference, because there is no shared capacity.

2.4.6 Which theory is correct?

All of these theories can be both supported by some data and refuted by others. At times, one of the bottleneck hypotheses has seemed to prevail, whereas other evidence has favored the unitary and multiple resource hypothesis.

Recently, Meyer & Kieras (1997a) have suggested the "radical" idea that there is no inherent cognitive bottleneck. Instead, they posit that performance decrements arise not from cognitive limitations, but rather from three sources. First, they can arise from structural (perceptual-motor) limitations. Second, dual-task decrements may be due to task instructions since it is known that task instructions can strongly bias subjects' performance (Gopher, 1993). Third, the dual-task decrement may be due to the performance strategies adopted by subjects. Task instructions can strongly bias subject to adopt an initial strategy, but during performance subject may explore alternative strategies, and their strategy selection may vary if the workload of the task varies.

Meyer and Kieras' hypothesis is unique from all the preceding dual-task interference hypotheses and theories in that it is the only one that has been and continues to be confirmed through detailed computational modeling in a veridical architecture, EPIC. Their hypothesis has demonstrated very good coverage of the empirical findings for the PRP task (Meyer & Kieras, 1997a, 1997b) and others.

2.5 Theoretical commitment of this work

As has been discussed, there are many theories that try to explain the cause of dual-task interference. There remains is no consensus in the community that any one of these theories is correct (or more correct) as they all address some, but not all, of the behavioral data. Yet to pursue modeling dual-task acquisition and performance, a selection and commitment to one of these theories of interference *must* be made.

In our work, we have adopted the same theories and commitments espoused by the creators of EPIC: there is no inherent cognitive bottleneck; the cognitive processor has unlimited computing capacity; structural limitations, selected strategies, and task instructions are responsible for the dual-task decrement; the key role played by the executive process in enabling dual-task performance.

Notes to Chapter 2

[1] In fact, an early mention of "executive routine" (Neisser, 1967) appears to have been inspired by the operation of computer systems.

CHAPTER 3

Selecting an appropriate computational architecture

The goal of this work is to develop a computational, task-independent framework for modeling the acquisition of the knowledge acquired during practice on a dual-task combination. To explore and develop this framework, we need a computational architecture—a fixed set of memories and processors with which a wide range of behaviors can be generated. How should one be selected? The approach followed here is first to identify architectural requirements as derived from the subgoals of the larger research goal and then select an architecture that satisfies all the requirements.

3.1 Architectural requirements

This work aims to:

- construct a model of a selected task
- identify and classify executive process knowledge
- develop a framework for the acquisition of executive process knowledge
- produce post-learning data that matches the observed data for the selected perceptual-motor task

These subgoals identify three components of the ideal computational architecture: a knowledge system with the right representations, a cognitive system that learns, and a psychologically-plausible performance system.

3.1.1 A knowledge system with the right representations

The first two subgoals call for a knowledge system that satisfies two requirements. The first is that the knowledge system can represent the common dichotomy of human knowledge: declarative and procedural knowledge.

Procedural knowledge is knowledge that is *impenetrable*; once acquired, it cannot be inspected by the cognitive system and hence cannot be reported. A typical example is the knowledge we use to ride bicycles. We all know how to ride bicycles. Yet we find we are unable to give a *complete* step-by-step procedure of how to successfully ride a bike. We can state how to get on and off the bike, and the many strategies for safe bike riding, but are unable to instruct the essence of bike riding; i.e., how to stay upright while pedaling.

In contrast, declarative knowledge is *reportable*. Propositions, facts, or statements, are examples declarative knowledge. An example is the instructions for changing the motor oil in one's car.

Any knowledge system used in modeling task acquisition and performance should also accommodate this knowledge distinction because various aspects of a task definition might, at least initially, be declaratively represented (such as task priorities, exceptions, the sequence of events as a task progresses such as when as stimuli will appear and disappear) while other aspects are procedural in nature, such as the steps/actions for actually performing the task.

Because we wish to identify and classify executive process knowledge, a second requirement of the knowledge system is that the knowledge be in an inspectable and understandable form. There are two general schools of knowledge representation: symbolic and numeric. Symbolic representations produce knowledge that is readable and understandable. The production rule is the possibly most prevalent representation for symbolic knowledge (Anderson, 1993; Forgy, 1981; Laird, Newell & Rosenbloom, 1987; Meyer and Kieras, 1997a; Newell, 1973; Schunn & Klahr, 1998). In the numeric community, knowledge is represented as a collection of numbers. A common example of this is the distributed weights of a connectionist network (McClelland, *et al.* 1986)). Given the need to inspect and understand the knowledge, a symbolic representation seems to be the appropriate choice.

3.1.2 A COGNITIVE SYSTEM THAT LEARNS

Because this work is about knowledge acquisition, only candidate architectures that have learning mechanisms can be considered. Additionally, since this work is focused on modeling dual-task performance, only architectures that have demonstrate some ability, or have some ability to support multiple concurrent threads of processing can be considered.

3.1.3 A PSYCHOLOGICALLY PLAUSIBLE PERFORMANCE SYSTEM

A performance system is needed to interface between the world and the learning and knowledge systems. The systems must therefore be able maintain a two-way dialogue representing the sensory and motor components in the human performance system. Our objective is to perform within a dynamic environment, therefore the performance system must be responsive to the changes in the world and must also be able to affect such changes.

The objective of this work is also to model human performance on a real task and to match actual performance data. It is therefore necessary that this system provide a psychologically plausible account of perceptual and motor abilities and limitations. For example, the visual perception system must account for limitations such as the varying quality of a stimulus that is a function of its eccentricity from the center of the retina, the time needed to move the eye to a location, or the ability of the eye to involuntarily track a moving target. In the motor subsystem, it must account for behavioral regularities such as the time to make a directed movement, also known as Fitts' law, the creation of motor features, and proprioceptive feedback, among others.

Finally the systems must posit some grammar for communication. Perception provides information to cognition, cognition provides motor commands to the motor system, and the motor system returns proprioceptive information back to cognition. It is important that this communication be at the appropriate level of abstraction, which is dependent on where one draws the line between perception, cognition, and action. It might be inappropriate, for example, for the visual system to send to cognition a bitmapped representation of the letter 'A' that just appeared on a screen, or for cognition to provide a list of individual muscle movements for pressing the 'A' key as the response.

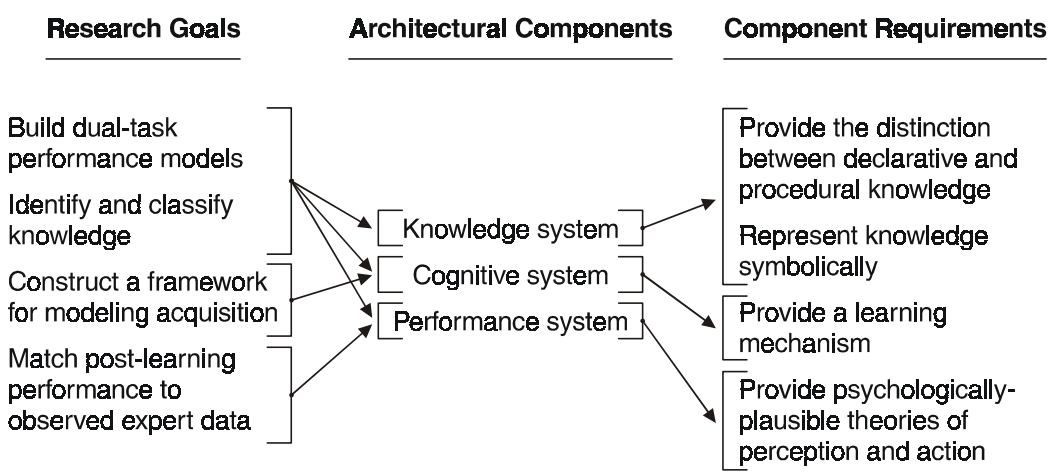


Figure 3.1 Derivation of the architectural requirements

3.2 Candidate architectures

Based on the subgoals of the overall research goal, three architectural components have been identified along with requirements of each component. They are summarized in Figure 3.1.

Given these components and requirements, one must now find an architecture that addresses all of them. Though there are many candidate architectures, only two are considered here. (In Chapter 10, we discuss the applicability of other architectures.) They are EPIC (Meyer & Kieras, 1997a, 1997b) and Soar (Laird, Newell, & Rosenbloom, 1987).

3.3 EPIC

EPIC (Executive Process-Interactive Control) is an architecture that was designed primarily to develop detailed accounts of human dual-task performance. It extends the work begun with the Model Human Processor (Card, Moran, & Newell, 1983). Like the Model Human Processor, EPIC consists of a collection of processors and memories. EPIC is distinguished from MHP in many ways, but two are most significant. First, the EPIC processors and memories are more elaborate, each representing a synthesis of much of the empirical evidence describing psychological phenomena. Second, EPIC is a system that can be programmed and executed. EPIC is written in Common Lisp and, at its core, is an event-driven simulation where these processors and a simulated task environment generate the events that drive the simulation.

EPIC has three classes of processors: perceptual, cognitive, and motor. See Figure 3.2 for a diagram of EPIC.

3.3.1 Perceptual processors

There are three kinds of perceptual processors: visual, auditory and tactile. These processors receive inputs from simulated sensors. The outputs of these processors are sent to the cognitive processor in the form of symbolic messages that are stored in working memory.

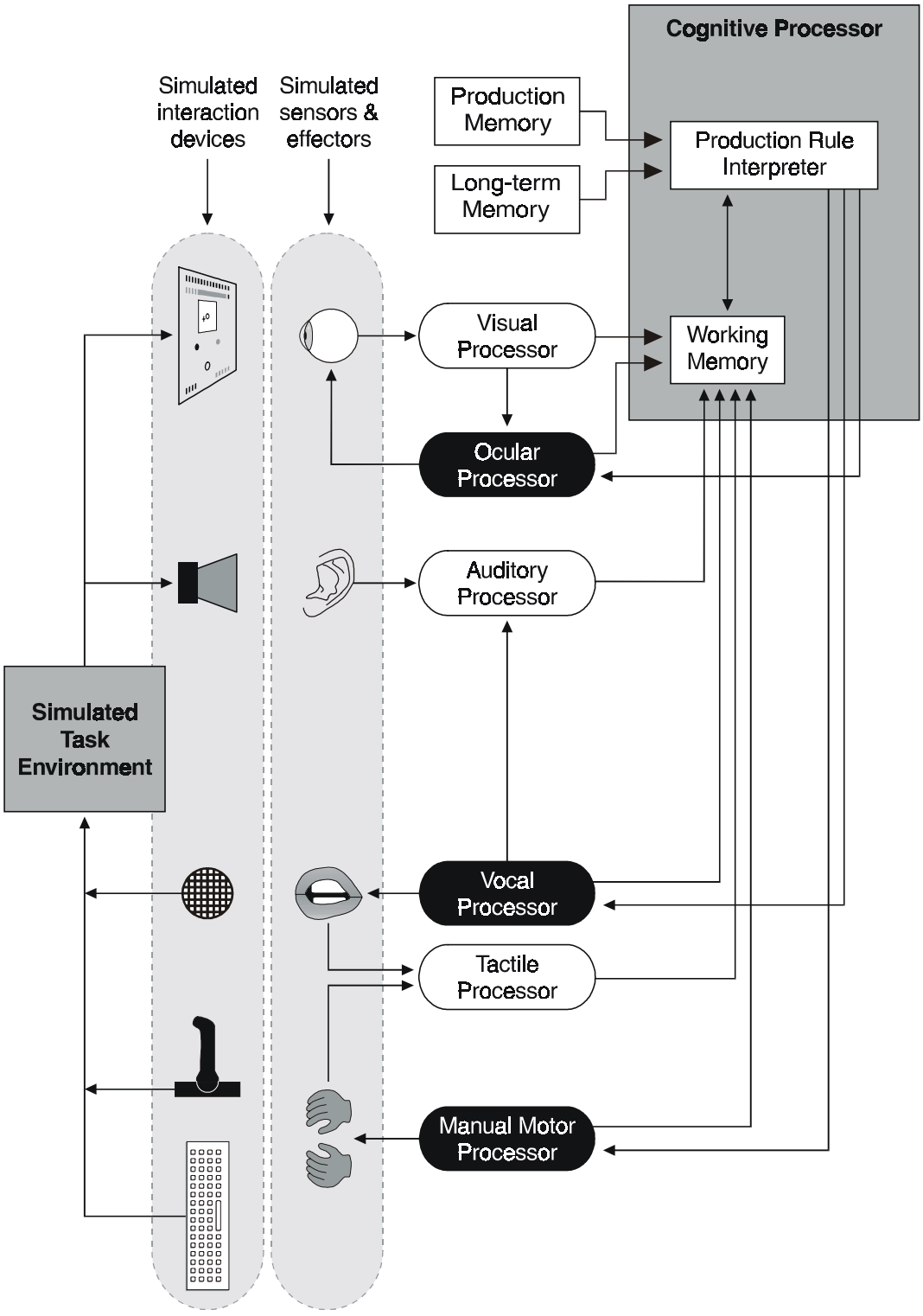


Figure 3.2 The EPIC human performance architecture

3.3.2 Cognitive processor

The cognitive processor consists of a working memory and a production rule interpreter. The interpreter accesses declarative knowledge from working memory and long-term memory, and procedural knowledge from a production memory. The cognitive system is a multi-match, multi-fire production system, meaning that all the rules that match during a match-fire cycle are allowed to fire. This is in contrast to many rule-based systems like OPS5 (Forgy, 1981) and ACT-R (Anderson, 1993), which are both multi-match, single-fire systems.

Upon receiving input from the perceptual processors, the cognitive processor fires all rules that match due to these inputs and the content of working memory. The firing rules produce changes to working memory and/or generate commands to be sent to the motor processors.

3.3.3 Motor processors

EPIC has three kinds of motor processors: ocular, manual, and vocal, of which the ocular and manual are the most developed. Motor processors accept commands from rules that have fired in the cognitive processor. When a command rule has fired, the command is immediately passed to the appropriate motor processor for processing and eventual execution.

EPIC's motor processors consist of two serial, semi-independent, and time-consuming phases, as depicted in Figure 3.3. The first is called the *preparation phase* and is followed by the *execution phase*. This two-phase construction agrees with empirical work on the distinction between movement preparation and movement execution (Abrams & Jonides, 1988; Gordon & Meyer, 1985; Rosenbaum, 1980; Sternberg *et. al.* 1978).

When a command is sent to a motor processor, the command is converted to movement features that describe the desired movement. These features are prepared serially. This phase operationalizes the findings by Rosenbaum (1980) of motor feature generation and preparation.

After the movement features have been created, they are "handed off" to the execution phase where the command is executed by simulated effectors. EPIC simulates the time necessary to execute the desired movement using long-standing predictors such as Fitts law (Fitts, 1954) and Meyer's law (Meyer, Abrams & Wright, 1990) for the case of a aimed manual motor command, and more recent time constants, for example the speed of a saccade; a saccade, is defined by Rosenbaum (1991), as "the eye 'jumps' that occur in tasks such as reading."

One of EPIC's motor commands is called `(PREPARE <action>)`. This command allows actions to be prepared but not executed. This is called *anticipatory motor programming*. Figure 3.4 depicts the activity in the two phases of the motor processor and demonstrates the performance benefits of using this prepare command. Consider a simple reaction-time task: "press the button

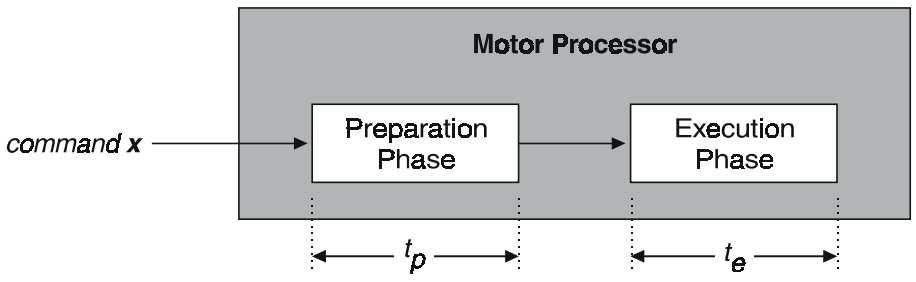


Figure 3.3 The structure of the EPIC motor processors

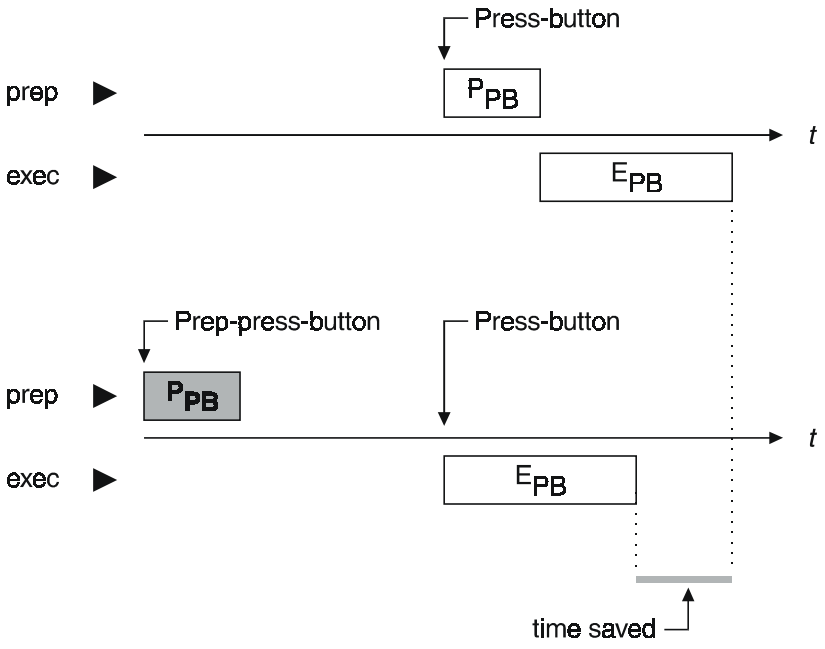


Figure 3.4 Comparison of non-prepared and prepared motor commands

when you hear the beep". The top diagram shows the non-prepared behavior. The rule `press-button` fires when the stimulus appears; the action is prepared (P_{PB}), and execution follows (E_{PB}). The bottom diagram shows the prepared behavior. A new rule, `prep-press-button`, which fires before the stimulus appears, prepares the movement features for pressing the button (P_{PB}). When the stimulus appears, the same `press-button` rule as before fires. Since the movement features have already been prepared, the movement can be immediately be executed. The difference in reaction times is shown and equals the preparation time, (P_{PB}). This prepare command allows EPIC models to account for the large effect of a precue stimulus on task performance.

However, anticipatory motor programming is most beneficial for commands where there is some certainty (greater than chance) that a particular motor action that will be required. For example, in the simple reaction-time task, prior to the stimulus appearance, it is known that the action will be to press the button. However, in a two-choice reaction-time task, there can be no such certainty prior to the stimulus appearance. The target button (assuming one of two buttons is to be pressed) depends on the identity of the stimulus, which cannot be known until it has appeared and been identified. Hence there is little benefit in preparing under these kinds of task conditions[1].

Motor processors not only receive input messages, but they also produce output messages that are sent to the cognitive processor for storage in working memory. These are status messages that inform cognition as to the state of the preparation phase, the execution phase, and the overall state of each motor processor. For example, when a command is sent to a motor processor, the processor returns the following messages to cognition: `MODALITY BUSY`, `PROCESSOR BUSY` and `PREPARATION BUSY`. When the preparation phase is finished `PREPARATION FREE` is returned. The prepared movement features can be handed off for execution *if and only if* the execution phase is free; i.e. not still executing a previous command. The processor returns `PROCESSOR FREE` to signal this condition. When the features are transferred for execution, `EXECUTION BUSY` is returned. When

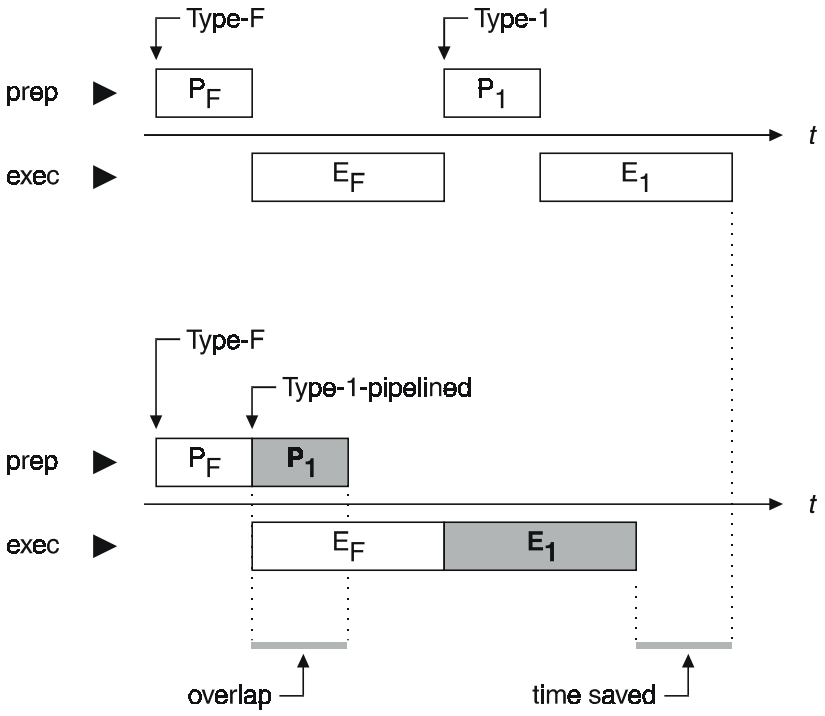


Figure 3.5 Comparison of non-pipelined and pipelined motor commands

the execution of the command has completed, `EXECUTION FREE` and `MODALITY FREE` are returned.

While all motor processors produce status messages, the manual motor processor returns additional messages which reflect the tactile or proprioceptive status of the command. For example, in the case of a keypress (a `PUNCH` command), the tactile processor returns the messages `STARTED PUNCH LEFT INDEX`, `DONE PUNCH LEFT INDEX`, `FINISHED PUNCH LEFT INDEX`, which can be interpreted as the following events: `key-touched`, `key-depressed`, `key-released`. In essence, these messages give status information that is one level of detail finer than the motor status messages.

The preparation and execution phases of the motor processors run semi-independently of each other. This means that it is possible to be executing one command while the preparation phase is preparing the movement features for the next command to be executed. The overlapping of the processing of two different commands in the motor processor is a technique known as *pipelining* and is ubiquitous in the design of modern microprocessors (Alexandridis, 1984; Hwang & Briggs, 1984; Foster & Iberall, 1985; Wilcox, 1987). This technique provides marked improvement in performance through maximal utilization of the processor's stages.

Figure 3.5 demonstrates this concept of pipelined commands and how it can improve performance. Consider the trivial task of typing the abbreviation for *Formula One*—an 'F' followed by a '1'. In the figure, `Type-F` and `Type-1` represent fired rules that generate the manual motor commands to type the letters 'F' and '1' respectively. (These rules assume that the CapsLock has been engaged).

The top diagram of the figure shows the non-pipelined approach. We see the `Type-F` command has fired, then the preparation of the movement features (P_F), then the execution of the command (E_F). The `Type-1` rule then fires and its features are similarly prepared (P_1), followed by the execution of the command (E_1). We see that there is no overlap of the commands in the processor; the `Type-1` rule does not fire until the 'F' key has been typed. In this case, the `Type-1` rule has been written such that it will fire only after the `EXECUTION FREE` message has been returned by the manual motor processor.

In contrast, the bottom diagram of the figure shows the pipelined approach. The `Type-F` rule fires and command processing proceeds as it did before. The difference in this approach though is the behavior of the `Type-1-pipelined` rule. Here, `Type-1-pipelined` is written such that it will match and fire when the preparation of the 'F' command is finished *and* when the features can be handed off for execution; i.e., when the `PROCESSOR FREE` message has been returned by the motor processor. The resulting P_1/E_F overlap in the processor causes the overall execution time of the task to be shorter than of the non-pipelined approach and this difference will be equivalent to the time to do P_1. Later in this thesis, we will see this technique applied in several instances to produce improvements in the performance of dual-task models.

3.3.4 Simulated task environment

Before a task model can be built in EPIC, a simulation of the task under study must be created; realized as a Lisp program. This simulation runs asynchronously with the other components (the perceptual, cognitive, and motor processors) of the EPIC architecture. The perceptual and motor processors interact with the task simulation to perceive and to change to the world. This simulation must be written in Lisp and like the EPIC processors, it must be written in the form of a process that is part of an event-driven simulation. The simulation must report on perceptual events—the onset, offset, movement, changes, such as shape, color, or pitch, of task objects—to the perceptual processors. It must also simulate the input devices (such as keyboards, mice, or joysticks) and how manipulation of these devices will affect the task (such as when moving a joystick causes a cursor to move, or pressing a button causes an auditory alert to cease).

3.3.5 Executive processes

Multiple-task models in EPIC generally incorporate an executive process. As discussed earlier, the purpose of the executive is to coordinate the progress of the tasks in the model. The EPIC executive process is encoded as production rules and stored in production memory with the rules for the individual tasks. In contrast to architectural control mechanisms—such as the Supervisory Attentional System (Norman & Shallice, 1986)—EPIC's formulation of the executive as regular productions results in a simpler and more homogeneous architecture.

Meyer & Kieras (1997a) list additional properties of executive processes used in their models:

- The executive only modulates the activity of tasks, by disabling and enabling tasks as needed for efficient performance.
- The executive rules do not contain procedural knowledge sufficient to perform any individual task.
- The executive can send motor commands that *enable* the future execution of tasks. Consider the case where two tasks are spatially separated, and both use visual stimuli. It is the responsibility of the executive to move the eye *between* the two task stimuli to bring them into view so that the associated tasks can be performed. An example of such a task will be seen in Chapter 4 and the use of such an executive rule will be seen in Chapters 5 through 8.
- The executive cannot modify the individual tasks' production rules.[2]
- The executive rule set for one dual-task combination may be task-specific, requiring that a

different rule set be developed to control a different dual-task combination.

- The fact that the executive rule set and the rule sets for the individual tasks are non-overlapping allow the rule sets for individual tasks to be reused to model other dual-task combinations.

3.4 Soar

The second candidate architecture for consideration is Soar. Soar is a general architecture for building artificially intelligent systems and for modeling human behavior (Laird, Newell & Rosenbloom, 1987; Laird, Rosenbloom & Newell, 1984; Rosenbloom, Laird, & Newell, 1993; Newell, 1990). Soar has been used to model central human capabilities such as learning, problem solving, planning, search, natural language and HCI tasks. Soar can be viewed as the union of a production system and a goal-oriented architecture.

3.4.1 Soar as a production system

As is the case with typical production systems, Soar contains two forms of memory—a declarative working memory, and a procedural production rule memory. Soar departs from typical production systems by incorporating a multi-match, multi-fire production rule interpreter. The production rule interpreter matches production rules against the contents of working memory. All rules that have matched are fired.

Soar has a rudimentary means of access and controlling the "outside world". Soar's working memory contains to specialized partitions. The first is for receiving input from programs external to Soar. The second partition is for providing commands for the world. Commands are placed in the output partition by production rules.

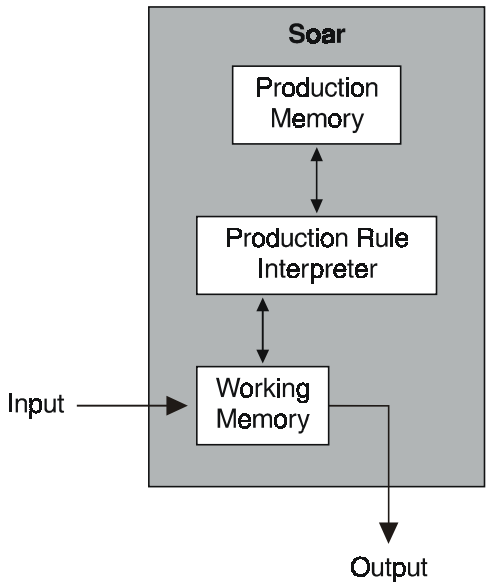


Figure 3.6 A simplified block diagram of Soar

3.4.2 Soar as a goal-oriented architecture

In Soar, an *operator* is a collection of rules that together implement a step in the decomposition of a task.[3] All Soar production rules pertain to directly or indirectly to either the proposal, selection (in the case where several have been proposed), implementation (performing the actions of the operator), or the termination of operators. When the knowledge encoded as productions is insufficient to either propose, select, or implement an operator, the system is said to be at an *impasse*. Hence, there are three main kinds of impasses: proposal impasses, selection impasses, or implementation impasses.

When an impasse occurs, an impasse-specific subgoal is automatically generated by the architecture. In the subgoal, the basic behavior of proposing, selecting, implementing, and terminating operators recurs, but in the subgoal, the activity is directed towards resolving the impasse. It is possible that the activity (or lack of activity) in the subgoal will itself result in yet another architecturally generated subogal.

A subgoal is automatically removed if the activity within the subgoal results in the resolution of the supergoal impasse. The change must be relevant to the type of impasse. Therefore, a proposal impasse is resolved when the subgoal activity proposes an operator for the supergoal; a selection subgoal is resolved when the subgoal activity selects one of the proposed operators in the supergoal; and an implementation impasse is resolved when the subgoal activity produces a change in the supergoal that implements the activity of the supergoal operator. When the impasse at a certain goal level has been resolved, activity at that goal level can continue.

3.4.3 Learning in Soar

See Figure 3.6 for a diagram of Soar.[4] It shows that Soar has basically the same components of the EPIC cognitive processor. The most significant difference is the *double-headed arrow* between production memory and the production rule interpreter. This signifies Soar's learning ability; the interpreter matches and fires productions in production memory, and it can create new productions that are stored in production memory.

Soar incorporates a *single* architectural learning mechanism called *chunking* (Laird, Rosenbloom & Newell, 1984). Soar is in sharp contrast with ACT-R (Anderson, 1993) along the dimension of the number of architectural learning mechanisms. ACT-R incorporates several learning mechanisms: base-level activation learning, associative strength learning; tuning of production rule strength, tuning of production rule probability and cost; and production rule creation through analogical compilation. By its commitment to a single architectural learning mechanism, Soar's chunking mechanism can be though of as a universal "store" command. It is believed that more complex, higher-level learning procedures are the product of chunking over complex problem solving, as demonstrated for inductive learning by SCA and SCA2 (Miller, 1993; Miller & Laird, 1996; Pearson, 1996).

Soar's chunking produces new production rules by summarizing the activity that transpired to resolve an impasse; i.e. it compiles the problem-solving that occurred in the subgoal and creates new productions. Therefore, a prerequisite to learning is the existence of an impasse, a subgoal, and the knowledge to resolve the impasse.

Typically, impasses and subgoals, and therefore learning, occur as a natural part of performing a task. On occasions where an impasse and subgoal does not naturally follow, productions can deliberately cause an impasse by generating additional operators, thus providing a problem-solving situation from which the desired knowledge can be learned. For instance, the learning procedures presented in this work are examples of this deliberate behavior. The procedures first force an impasse and a subgoal, at which point the procedures perform problem solving in the subgoal. The product of the problem solving is returned to the supergoal, which causes a chunk to be built and the resolution of the impasse.

In combination with various problem solving methods, chunking has been found sufficient for learning in a wide variety of applications (Altmann, 1996; Huffman, 1994; Lewis, *et al.*, 1990; Miller and Laird, 1996; Pearson, 1996; Rogers, 1997; Rosenbloom & Newell, 1993; Steier, *et al.*, 1987).

3.5 Evaluating the candidate architectures

It is clear that neither of the two candidate architectures satisfy all the requirements set forth in the beginning of this chapter. They both are symbolic computational architectures that capture the procedural versus declarative knowledge distinction in their representation systems. However, only Soar possesses a learning mechanism while only EPIC has psychologically-plausible perception and action mechanisms.

3.6 EPIC-Soar: A hybrid architecture for learning and performance

From the descriptions of EPIC and Soar, it was clear that it could be fruitful to combine the two systems. EPIC possesses the most thorough computational theory of perceptual and motor processors. EPIC's cognitive processor is a multi-match, multi-fire production system. The EPIC architecture as a whole was designed to explore multiple-task behavior. EPIC's cognitive processor does not presently learn however. While EPIC can model novice and expert behavior, its present lack of learning precludes modeling the transition from novice to expert behavior.

Soar, represents one of the most complete computational theories of human cognition. It makes a parsimonious commitment to a single mechanism for learning. It also supports the knowledge representation requirements. Learning, search, problem solving, planning, among others, are capabilities that can be demonstrated in Soar. Soar also have a multi-match, multi-fire production system and has been applied to multiple-task situations (Aasman, 1995; Covrigaru, 1992). However, Soar's theories of sensory or motor processes are not as mature as those for its cognitive processor.

The potential for a synergistic merging of EPIC with Soar was obvious and the endeavor was undertaken. The resulting hybrid architecture has been called EPIC-Soar (Chong, 1995; Chong & Laird, 1997, Chong, 1998a, 1998b). EPIC-Soar represents a parsimonious integration of the perceptual and motor processors of EPIC with Soar. This merger is an attempt to get both the detailed predictions and explanations provided by the perceptual and motor processors of EPIC (capabilities that Soar does not possess) and the problem solving, planning, and learning capabilities of Soar (capabilities that EPIC does not possess). See Figure 3.7 for a diagram of EPIC-Soar.

3.6.1 Technical details of EPIC-Soar

In EPIC, the cognitive processor accepts perceptual and motor messages as input and it generates motor commands as output. Because of this modular design, combining the orthogonal aspects of EPIC and Soar was relatively easily accomplished. We have essentially performed a brain transplant; EPIC's cognitive processor has been "removed" and Soar "put" in its place. The substitution of Soar for its native cognitive processor is transparent to EPIC.

EPIC is written in Common Lisp and Soar is written in C, therefore the two systems had to remain as two physically distinct entities. The connection between the two systems was accomplished by the use of UNIX sockets. Through this mechanism, perceptual/motor messages, which would normally be sent to EPIC's native cognitive processor, are intercepted and sent to Soar as input to its working memory. Soar then cogitates on the input and outputs motor processor commands to EPIC using the socket.

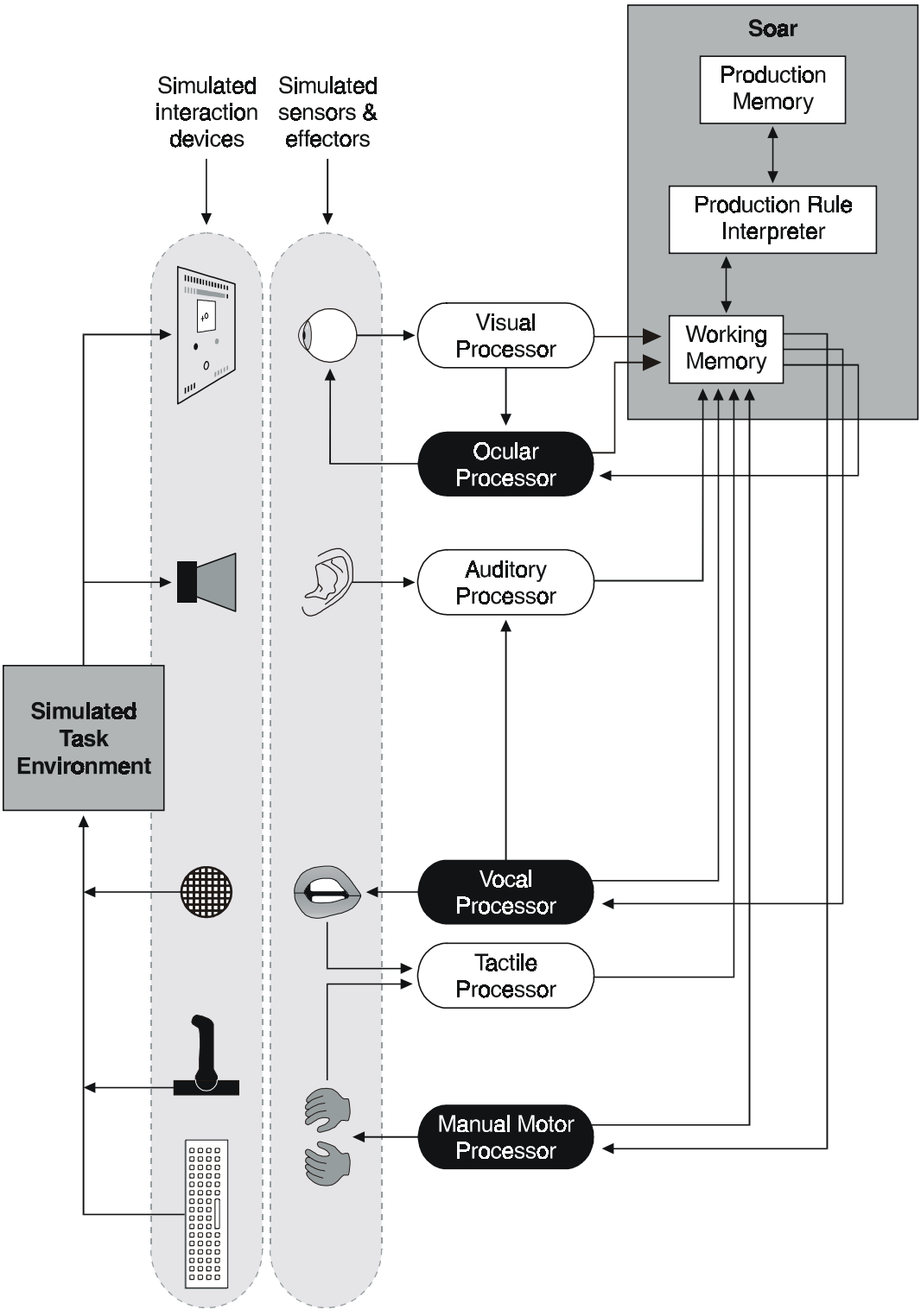


Figure 3.7 Block diagram of EPIC-Soar

The details of the cycle of interaction and information exchange between the systems is as follows: EPIC perceives the world and its motor system (tactile/proprioceptive feedback) and sends these perceptual messages to Soar; EPIC then waits for a response from Soar; Soar accepts the inputs from EPIC; Soar runs for one cognitive cycle and returns to EPIC any motor commands that may have been generated; Soar then waits for inputs from EPIC; EPIC accepts the motor commands, if any, that Soar has sent and executes them. This cycle, which represents 50ms of simulated time, is repeated over and over.

Because EPIC is an event-driven simulation, an important and necessary engineering consideration was to force EPIC-Soar to operate in a lock-step fashion. This had the benefit that Soar's decision cycle trace would match the output of the EPIC decision cycle trace. This synchronization significantly helped in debugging the EPIC-Soar architecture and the models that were later developed in EPIC-Soar.

In the later chapters of this thesis, there will be much discussion about the development of models and procedures. To make the implicit explicit, *all* cognitive (high-level) production-based task modeling is done in Soar, while EPIC's perceptual and motor processors can be thought of as modeling the low-level aspects of performance—the limitations of perception and its transmission, the creation of movement features and their execution, to name a few.

Notes to Chapter 3

[1] Some benefit can indeed be gained if one was to always prepare for one of the possible actions. Assuming a two-choice task, an improvement would be had on 50% of the trials. Though this is a reasonable strategy, it *might* be the case that subjects do not use it since it results in higher error rates in responding. Evidence supporting this hypothesis comes from a study by Gordon & Meyer (1987). They used a two-choice reaction time task where the subject was always to be prepared to produce a primary response although there was an equal likelihood that the secondary response would be required. They reported that the errors rates for the unprepared secondary response were higher than for the prepared primary response. The RT for the secondary responses were higher than the primary responses, thus ruling out a speed-accuracy trade-off explanation of the findings. The differences in both the error rates and RT was found to be statistically significant. Therefore, it appears that higher error rates can result from being prepared to produce one of n responses when all n responses are equally likely.

[2] The EPIC executive processes comply with this property by default since EPIC does not presently learn.

[3] Soar is unique among productions systems in that its productions are of two types: those whose actions are persistent, and those whose actions are not. Persistent actions can only be generated by rules that implement operators. In contrast, all actions of EPIC rules (as is true for the majority of productions systems) are persistent.

[4] Figure 3.6 only depicts Soar along the dimension of memories and the production rule interpreter. There are other dimensions which along which Soar can be illustrated that do it justice. This illustration was used because is allows Soar to be easily compared to EPIC's cognitive processor.

CHAPTER 4

The Wickens task

As stated in Chapter 1, this work focuses exclusively on perceptual-motor tasks; i.e. tasks where perceived events in the world require overt actions in the world. This is in contrast to purely cognitive tasks such as memory search or counting.

Martin-Emerson & Wickens (1992) performed a dual-task study to evaluate the effect of vertical separation of a choice task on the performance of a tracking task. Their task environment is shown in Figure 4.1. The task consisted of a continuous tracking task and a choice-reaction time (CRT) task. The tracking-task performance was recorded as the average RMS error (distance between the cursor and the center of the target). The choice-task performance was recorded as average reaction-time, in milliseconds. The application of this study is to the design of the heads-up displays used primarily in aviation. This task, which we have called the *Wickens task*, was chosen for our work primarily because a model of this task already existed in EPIC (see Chapter 5).

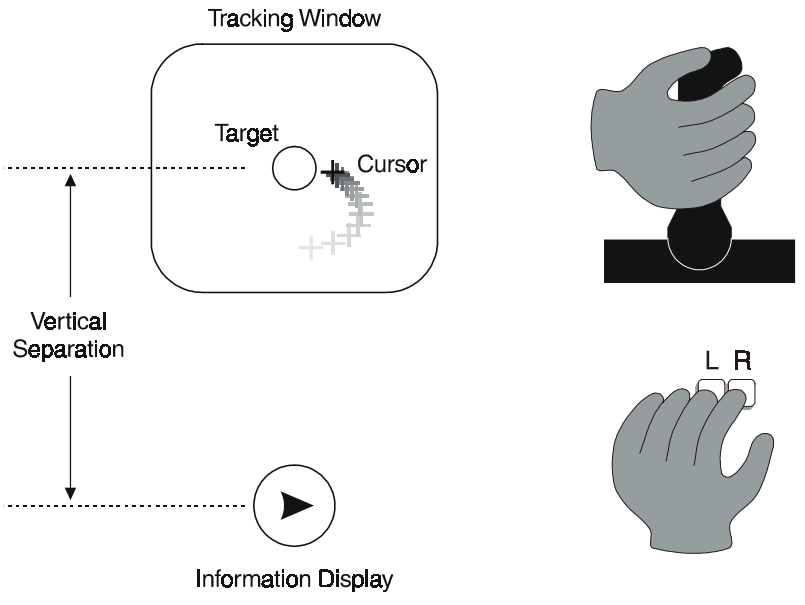


Figure 4.1 The Wickens task environment.

The experimental setup included two displays: a tracking window and an information display. The tracking window contained a cursor and a target circle. In the tracking task, the subject used a joystick to keep a cursor (which is always moving) in the target circle. The joystick was controlled by the right hand. This task simulated a pilot tracking a ground target (such as the end of a runway) when landing.

The choice-reaction task stimulus was presented in the information display, where either a left or right arrow would periodically appear. The stimulus duration was one second. When the stimulus appeared, the subject was to press one of two buttons beneath their middle and index fingers on the left hand; the left button for the left arrow, or the right button for the right arrow. This task simulated warnings or other indications that may appear while a pilot is landing the plane but require some sort of immediate response.

The task requirements were for the subject to keep the cursor in the target, but to respond to the choice stimulus on the information display as soon as possible. Average RMS tracking error, average reaction time, and response correctness were gathered. The tracking error was recorded only during the one-second presentation of the choice stimulus and the one-second immediately following the stimulus offset; a total of two seconds. It was anticipated that the most tracking errors would occur during this period due to the allocation of attention to the choice task.

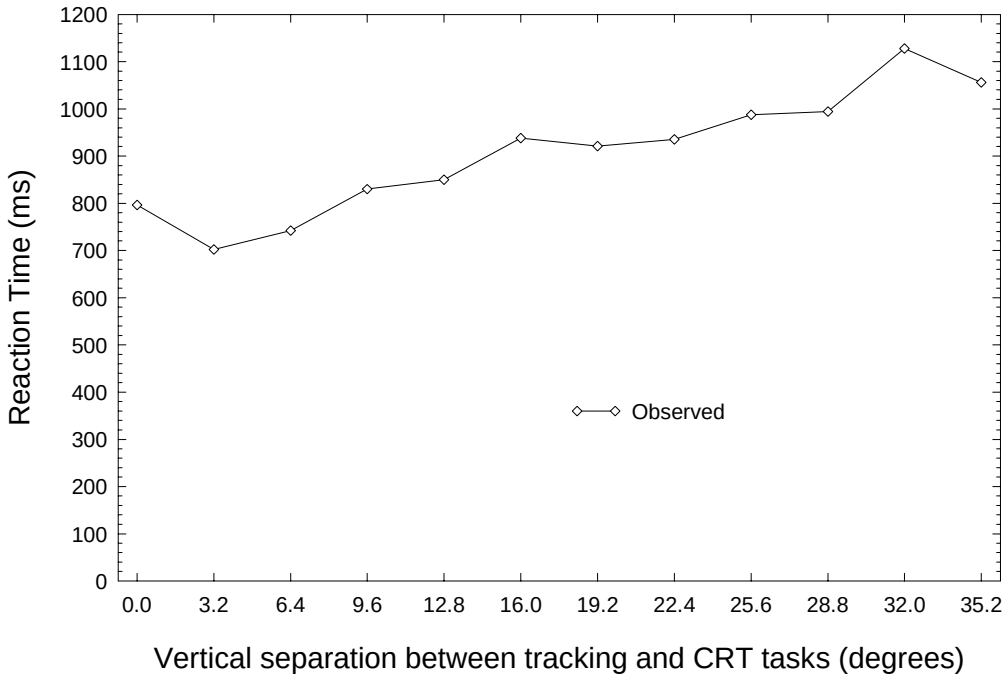
In the original study, several experimental variables were manipulated, but only two were considered in the modeling efforts here. The first was the vertical separation, measured in visual angle degrees, between the tracking and choice displays. Separations ranged from 0° (where the information display and the target were superimposed and centered, one over the other) up to 35.2° of visual angle, in increments of 3.2°.

The second manipulated variable was the difficulty of the tracking task. The cursor used in the tracking task was continuously perturbed. The high or low difficulty conditions differed by the bandwidth of the perturbation used to determine the cursor movement.

For our modeling work, we varied the vertical separation throughout the full range of separations, but fixed the tracking difficulty at "high". It was expected that this condition would put the most stress on the model and, as a result, make the effects of changes to the model most evident.

The task is performed as follows. On the first trial of a condition (vertical separation), the task objects—the cursor, target, and information display—are presented and the cursor immediately begins moving. The subject must use the joystick to minimize the tracking error; the distance between the meandering cursor and the target. At some bounded random time, while the tracking task is being performed, the stimulus for the choice task appears. Depending on the visibility of the choice stimulus, the subject might have to look away from the cursor to fixate on the stimulus so that it can be clearly seen. The subject must then respond to the stimulus as quickly and accurately as possible while continuing to perform the tracking task; i.e. keeping the tracking error low. The generation of a response to the stimulus marks the end of a trial. This is a continuous task however, so tracking continues until again, at some bounded random time, another choice stimulus appears, requiring that a response be generated. This continues until the desired number of CRT task trials have been completed.

Figure 4.2 shows the empirical human data for this task as reported in Martin-Emerson & Wickens (1992). The choice task, whose reaction-time performance is shown at the top, shows an effect due to the vertical separation. This effect is strongly related to the increasing saccade times (when saccading from the cursor to the choice stimulus) due to the increased vertical separations. The average RMS error rates of the tracking task showed a similar relationship, but the effect was much less pronounced. The tracking performance was the primary result in this study. It will likewise be the primary measure for our models although RTs will also be reported.



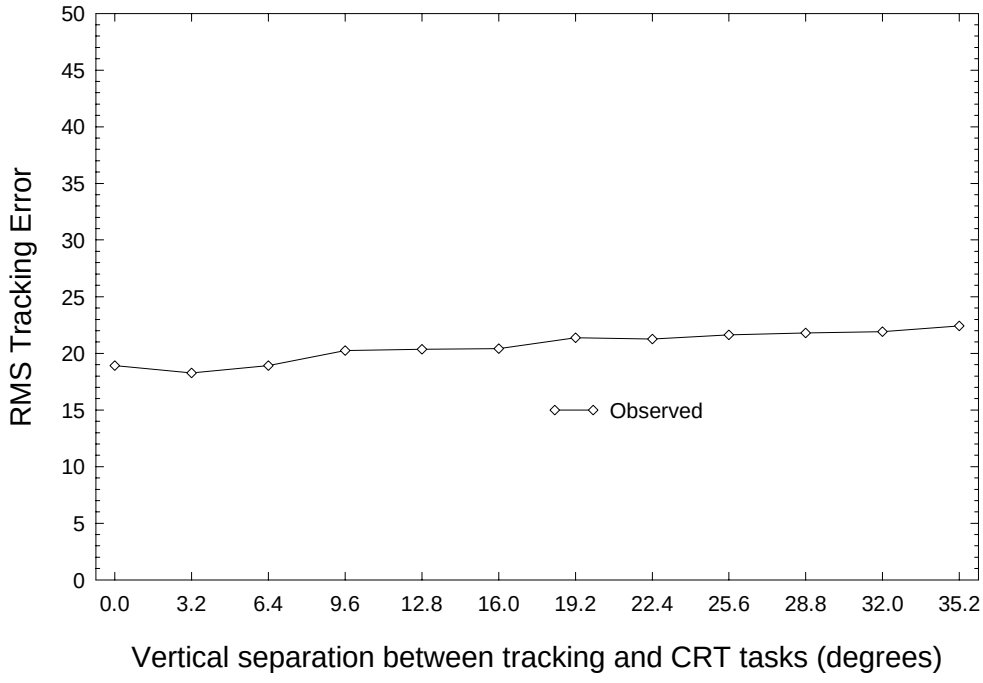


Figure 4.2 Expert human performance on the Wickens task

CHAPTER 5

EPIC on the Wickens task

EPIC has been used to produce a model for the Wickens task (Kieras, 1994; Kieras & Meyer, 1995a). The production rules of the EPIC model realize a model of *expert* dual-task performance for the task. As can be seen in Figure 5.1, the EPIC model provides a good overall match to the empirical data on both the reaction-time and tracking error measures; RMS error between the model and the observed data was 60.52 and 1.11, respectively.[1]

5.1 An execution trace of the EPIC model

Figure 5.2 shows a portion of a trace of the EPIC model as it runs one trial. This trace shows the cognitive cycles and the names of the production rules that fired during those cycles. Note that many cycles are missing in the trace. Many empty or inconsequential cognitive cycles have been removed in order to promote a general understanding of the model's behavior.

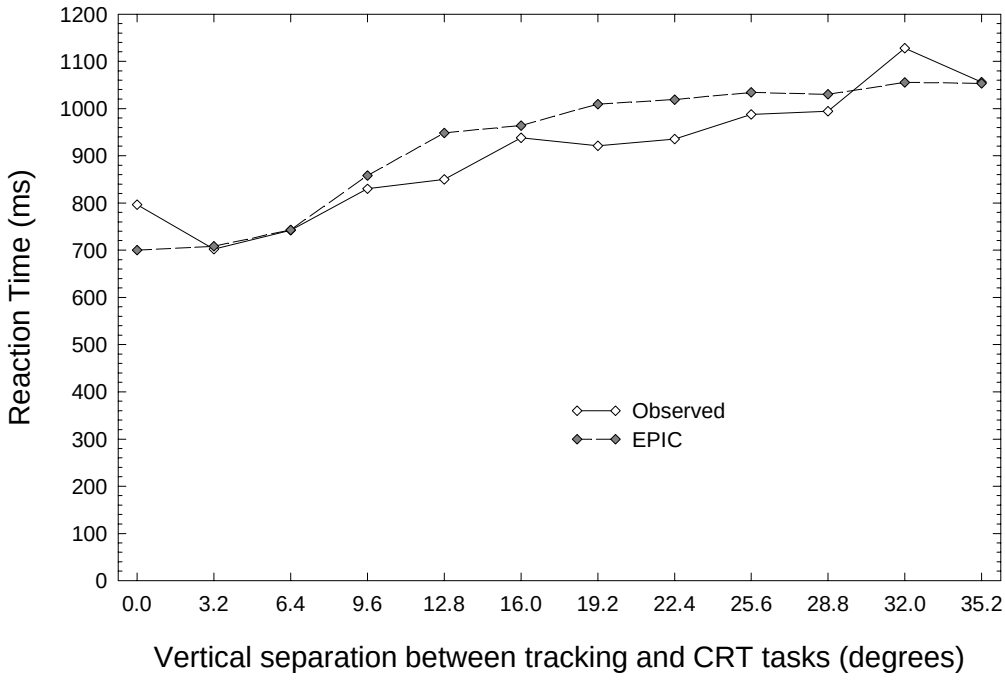
Note that the trace begins with cycle 20. During the first 19 cycles, EPIC identifies all the objects of the task—the cursor, target, and information display as defined in Figure 4.1. This identification phase is modeled realistically in that eye movements are performed to look around the task display to allow identification of the task objects. This identification phase was not shown since no data was gathered about this phase in the original study, and the performance on this phase does not impact the results.

At cycle 20, EPIC fires the main rule of the tracking task which sends a command to move the joystick to cause the cursor to move towards the target. This rule will fire often during the trace to keep the cursor on or near the target.

At cycle 21, EPIC fires an executive process rule that is an anticipatory motor programming rule. As presented in Chapter 3, the EPIC architecture has the ability to prepare the movement features for future action without producing the action. Figure 3.4 showed that preparing for a future action can save feature preparation time when the action is finally performed; in this case, at cycle 54. This rule is classed as an executive rule because it is not part of the knowledge sets for either the choice task or the tracking task.

At cycle 34, the eye is made to move to the cursor. Recall that the cursor is constantly moving, either due to its normal random perturbation or due to the joystick movements. In either case, it is necessary for the eye to be on the cursor so that an accurate evaluation of the error and direction of a corrective joystick movement can be made.

At cycle 37, the ocular system is again prepared to look at the choice stimulus location even though this was just performed at cycle 21. This re-preparation is necessary since the motor processor of EPIC can only hold the preparation features for one action at a time. At cycle 21, the features for looking at the choice stimulus location were stored. Then at cycle 34, a saccade was



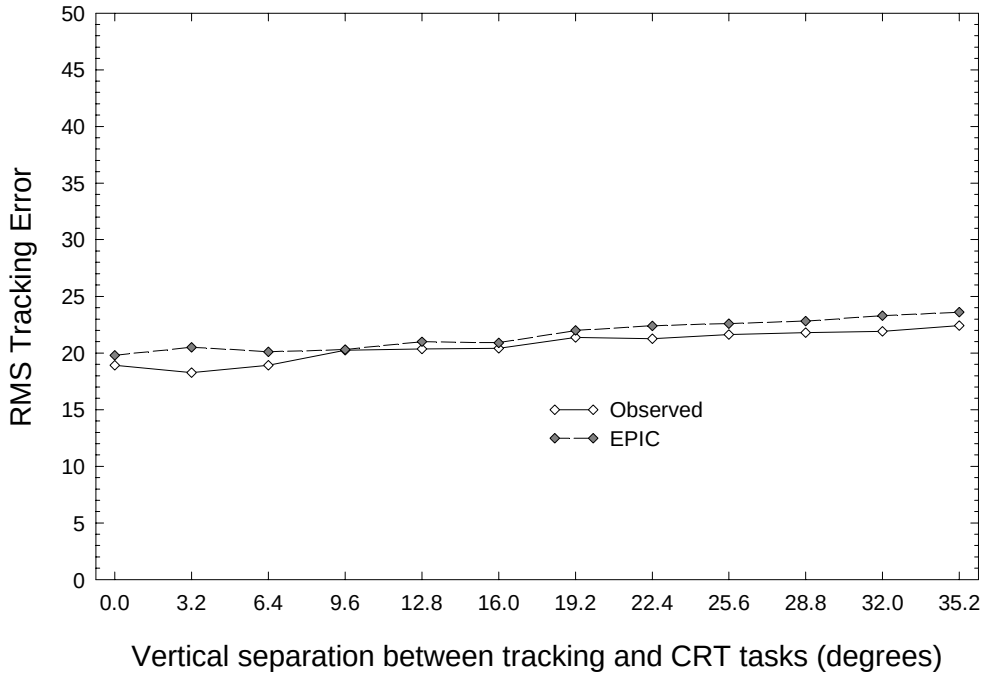


Figure 5.1 The EPIC model on the Wickens task.

```
20   TRACKING-TASK-move-cursor
21   EXECUTIVE-prepare-to-move-eye-to-choice-stimulus
28   TRACKING-TASK-move-cursor
32   TRACKING-TASK-move-cursor
34   TRACKING-TASK-watch-cursor
36   TRACKING-TASK-move-cursor
37   EXECUTIVE-prepare-to-move-eye-choice-stimulus
42   TRACKING-TASK-move-cursor
47   TRACKING-TASK-move-cursor
52   choice stimulus has onset
     TRACKING-TASK-move-cursor
54   EXECUTIVE-move-eye-to-choice-stimulus
55   EXECUTIVE-suspend-tracking-because-eye-is-away-from-cursor
     EXECUTIVE-put-eye-back-on-cursor-asap
60   TRACKING-TASK-watch-cursor
     EXECUTIVE-resume-tracking-since-eye-is-back-on-cursor
61   TRACKING-TASK-move-cursor
63   CHOICE-TASK-start
64   EXECUTIVE-prepare-for-choice-stimulus-onset
     CHOICE-TASK-nomatch-left-arrow
65   TRACKING-TASK-move-cursor
     CHOICE-TASK-match-right-arrow
66   EXECUTIVE-suspend-tracking-to-prevent-motor-conflict
67   CHOICE-TASK-send-response
68   EXECUTIVE-restart-tracking
71   TRACKING-TASK-move-cursor-asap
```

Figure 5.2 An abridged trace of the EPIC model on one trial of the Wickens task

commanded and the preparation of its movement features replaced the features that were prepared in cycle 21. Therefore, preparation is a *homeostatic* goal, that is, it must not only be achieved, but also be *maintained*.

At cycle 52, cognition becomes aware that the stimulus appeared in the task environment. Because EPIC accounts for the time for notification of this appearance to arrive in cognition (a delay of about 100 milliseconds or two cognitive cycles), we can know that the stimulus actually appeared in the environment at around cycle 50.

At cycle 54, in response to the onset, a rule is fired to move the eye from the cursor to the location of the choice stimulus. It is important to note that this rule only applies when the visual features of the stimulus are not sufficiently detailed. This situation occurs when distance between the cursor (which the subject has been looking at prior to the stimulus appearance) and the stimulus is sufficiently large. (Assuming a tracking model that is diligent in keeping the cursor at or near the target, then an eye movement would only be needed when the vertical separation condition was sufficiently large enough.)

At cycle 55, we see the first mediation by the executive process. On cycle 54, the eye was directed to the choice stimulus. However, performance on the tracking task requires that the location of the cursor be accurately known. Since this is not possible, the executive disables the tracking task to prevent erroneous tracking movements while the eye is away from the cursor.

Also on cycle 55, the eye is directed back to the cursor as soon as it is possible to do so. The reason for the urgency is so that the tracking task can be resumed before the cursor wanders too far from the target.

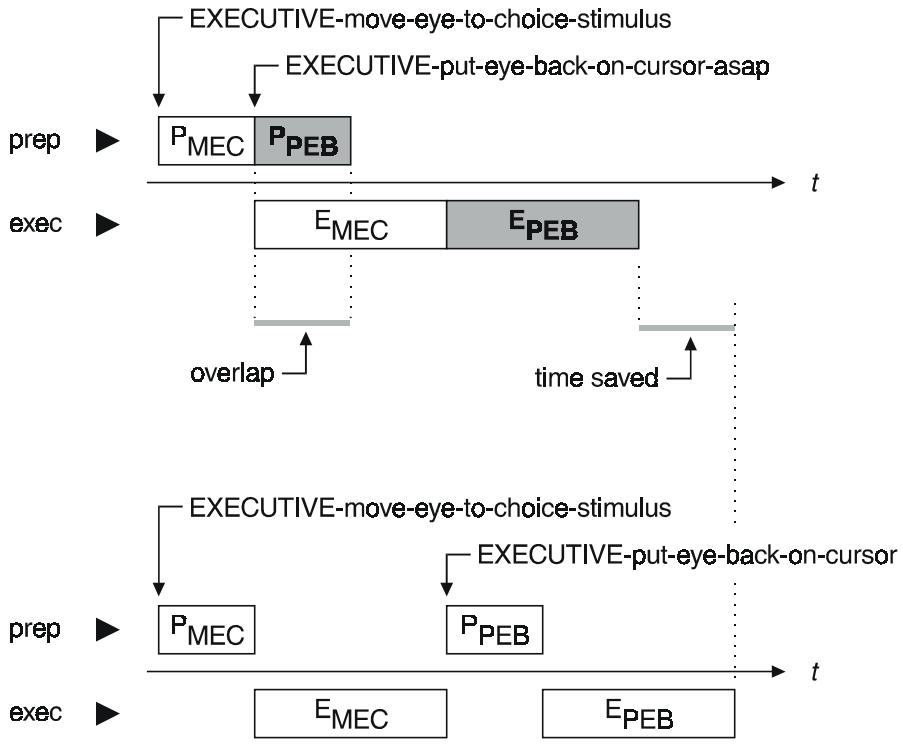


Figure 5.3 Overlapping execution and preparation of consecutive ocular commands

The "as soon as possible" aspect of the rule is implemented using the pipelining technique seen earlier in Figure 3.5. Figure 5.3 demonstrates how it is used in this model. The top diagram of the figure shows the preparation (P_{MEC}) followed by the execution (E_{MEC}) of the first command, `EXECUTIVE-move-eye-to-choice-stimulus`. Note that the `EXECUTIVE-put-eye-back-on-cursor-asap` rule fires as soon as the ocular motor processor's preparation phase has just finished preparing the first command. As a result, preparation P_{PEB} overlaps with execution E_{MEC}. The execution E_{PEB} must wait until the execution E_{MEC} is finished since the eyes can only be directed to one location at a time. This overlap in the motor processor can occur because the `EXECUTIVE-put-eye-back-on-cursor-asap` rule is preconditioned on `PROCESSOR FREE`, which occurs when the preparation phase has finished and the movement features can be handed off for execution. Because these phases are distinct, semi-independent, and sequential, it is possible for the motor processor to be executing one command while it is preparing the movement features for a subsequent command.

To again demonstrate how utilizing this overlap is beneficial, the lower diagram of the figure shows the behavior of a less urgent rule, say `EXECUTIVE-put-eye-back-on-cursor`. Such a rule would wait until the ocular motor processor was finished executing the previous command before it could fire. As is shown, this rule would fire after E_{MEC} has finished executing.

It is clear that `EXECUTIVE-put-eye-back-on-cursor` takes longer to finish since there is no overlap of processing. The time saved with the efficient approach is equal to the preparation time for P_{PEB}. This time is typically between 100 to 200 milliseconds. This time savings seems small but in the context of a dynamic task such as this one (where the cursor is constantly moving),

a 100 to 200 milliseconds savings can directly translate into better tracking error rates since the cursor will have been unattended for a shorter time. Also, recall that tracking error is only collected during the two seconds after the stimulus has onset. Therefore it is important to track as efficiently as possible during this period of time.

Returning to the trace, at cycle 60, we see that the tracking task has been resumed because the eye has returned to the cursor; cycle 55.

The choice task is finally begun at cycle 63. The reason for the nine cycle delay (awareness of the onset occurred back at cycle 54) in starting the task is in part due to the movement of the eye down to the stimulus. Once the eye had arrived at the stimulus, the rest of the delay is due to EPIC's modeling of the transmission of the stimulus features through sensation, perception, and finally into cognition.

At cycle 64, the model finds that the stimulus is not a left arrow. On the next cycle, it successfully identifies the stimulus as a right arrow. Also note that the tracking is going on while the choice task is being performed; cycle 65.

At cycle 66, we again see the executive process asserting itself to disable the tracking task. The reasoning is that because the choice stimulus was just identified, a manual motor response can be immediately generated. However, the tracking task also wants to use the manual motor processor. In order to prevent a manual motor conflict, the tracking task must be disabled.

At cycle 67, the choice task safely sends the response. After the response has been sent, the tracking task can be safely restarted in cycle 68.

The `TRACKING-TASK-move-cursor-asap` rule firing during cycle 71 represents the same utilization of the two-phase nature of the motor processors as described earlier. Analogous to the discussion of cycle 55, the rule here was designed to produce a joystick movement as soon as possible after the choice response had been made. Specifically, this rule sends a joystick command when the manual motor processor returns `PROCESSOR FREE`. The rational for this rule is that while the choice task was sending the response command, the tracking task was disabled from producing joystick commands. During this disabled period, the cursor drifts further and further from the target, increasing the tracking error. Hence, a solution is to produce a joystick command *as soon as possible* after the choice task response has been sent.

This rule differs from the normal tracking rule (`TRACKING-TASK-move-cursor`) in that it is designed to fire when the execution phase of the manual motor processor is free; an event which can occur quite some time after the preparation phase is free. Again, the pipelining technique is used. Figure 5.4 demonstrates this. The top diagram of the figure shows the behavior of the "asap" rule. This figure differs from Figure 5.3 in that there is an additional overlap of the execution phases. Unlike the previous figure, which shows ocular motor behavior, this figure shows *manual* motor behavior where the *single* manual motor processor can control the left and right hands—a key theoretical commitment of EPIC. Recall that the tracking task uses the right hand to control the joystick while the choice task responses are generated with the left hand. This figure shows that although the hands are commanded from a single manual motor processor, the hands themselves can simultaneously make different movements. The lower diagram of Figure 5.4 shows the behavior using the normal tracking rule.

As expected, the "asap" rule provides a large time savings, and as discussed before, this savings can directly reduce the tracking error. It is worthwhile to restate that tracking error is collected only for two seconds after the stimulus has onset. Therefore it is especially important to track as often as possible during this period of time.

5.2 Discussion

As Figure 5.1 shows, the EPIC model produces very good overall match to the empirical data. This success is a direct result from the combining of psychologically realistic perception and action

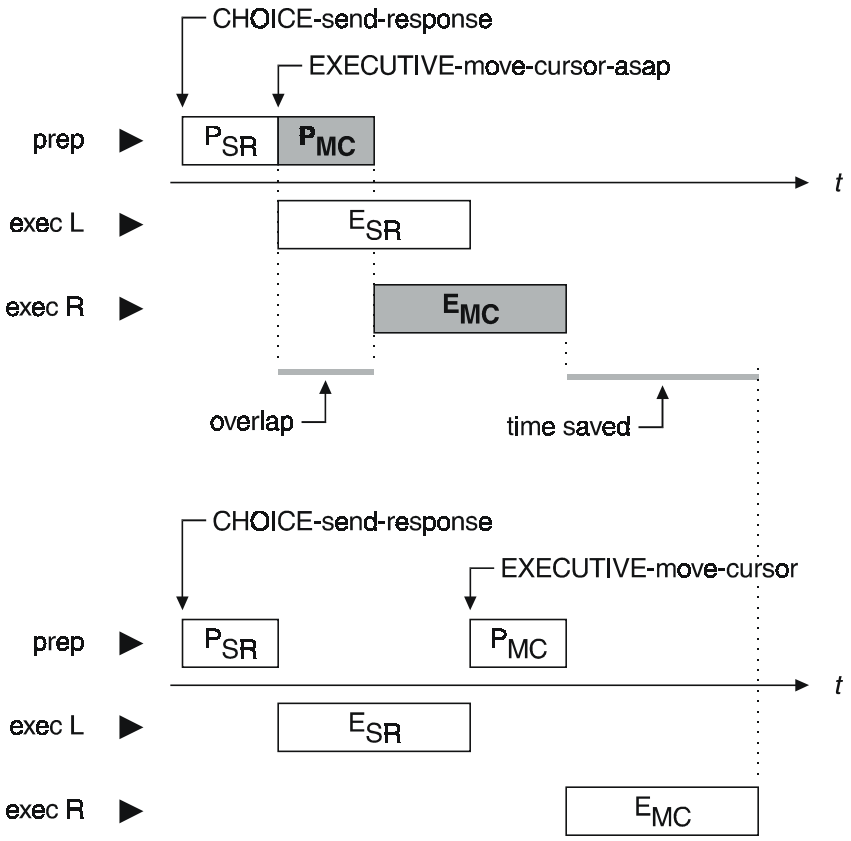


Figure 5.4 Overlapping execution and preparation of consecutive manual commands

with cognition. The following EPIC capabilities and attributes played a critical role in order to achieve this level of fit to the data:

- the two-phase construction of the motor processor;
- the ability to prepare the features for a future action;
- the ability to overlap two actions in the motor processor;
- modeling of the information fidelity of the visual sensory fields;
- realistic motor execution timings;
- a commitment to a single manual motor processor;
- realistic propagation delays from sensation, through perception, and to cognition;
- the use of an executive process for mediating the independent tasks.

Concerning the executive process, we saw that some of those rules were basically task facilitation rules that moved the eye between the tasks.

Other rules, however, could best be defined as implementing an explicit, task-specific strategy for coordinating the two tasks. Tasks are disabled and enabled for fundamental reasons (preventing motor conflicts) and for task performance reasons (disabling the tracking task because the eye was away from the cursor). In a sense, these rules "micro-manage" the execution of the tasks, providing

deliberate control of how the steps of each task are interleaved and as such, they are quite task specific.

Notes to Chapter 5

[1] The RMSE will be used to evaluate the goodness-of-fit of all expert models in this work. In addition to this measure, one would also want to know if the model's performance is significantly different from the observed data. However, we would need to know the variability of the human data, something that was not available at the time this work was reported.

CHAPTER 6

The classification of executive process knowledge

This chapter will present the first of three main phases of this work. The following two chapters will present the remaining phases. This first phase had four objectives. First, although a working performance model already existed in EPIC, we needed to construct an analogous model in EPIC-Soar since all the future work would be done in EPIC-Soar. Soar is a more elaborated theory of human cognition as compared to EPIC's cognitive processor; i.e. Soar has many more defining attributes in addition to being a multi-match, multi-fire production system. For example, Soar contains concepts such as problem spaces, a preference semantics for production rule actions, two forms of persistence for working memory elements, operators, architecturally-generated subgoaling, an architecturally-maintained goal stack, and an architectural learning mechanism. As a result, the construction of a Soar performance model was not just a syntactic transformation of the EPIC production rules; i.e. a mapping from one production system syntax into another. Additionally, because of these differences in complexity just outlined, the Soar model may make some aspects of the EPIC model less relevant while increasing the consideration of others.

Second, this phase was used to validate that the EPIC-Soar hybrid architecture was sufficient for modeling dual-task expert performance. If we discovered at this step that an adequate performance model (one that gave a good match to performance) could not be built, then there would be no point in continuing to search for and develop the desired knowledge acquisition framework using this architecture.

At the end of the previous chapter, some of the EPIC executive process rules were described as defining an explicit, *task-specific* strategy for controlling the interleaving of the two tasks. To create the EPIC-Soar model of the task, these control rules could have been adopted. However, a fundamental tenet of cognitive modeling in Soar (and presumably other learning architectures) is that the modeler must be able to articulate the source of *all* the knowledge that comprises the model. Since it is known that executive process knowledge is acquired with practice (as discussed in Chapter 2), we needed to propose the mechanism for acquiring the explicit control knowledge used in the EPIC model. Such a mechanism was not immediately apparent, therefore the EPIC executive process rules could not be added to our Soar model.

Alternatively, we wondered if perhaps a *minimalist*, *task-independent* executive process could be used while still producing an acceptable match to the expert data. Such a process would freely allow both tasks to run concurrently; i.e. without task control of any kind. One problem with this approach is that motor conflicts—or "jams", as we will refer to them—are possible; two tasks simultaneously sending commands to the same motor processor.

When this a jam occurs, a task-independent conflict resolution procedure would intervene and allow the preferred command (as specified or implied by the task instructions) to be sent. As a by-product, a conflict-avoidance rule are learned for preventing future conflicts between the same commands. In this approach, the executive process initially consists of a task-independent conflict resolution learning procedure. As the task is performed and conflicts arise, this procedure resolves the conflicts and learns new rules for avoiding the same conflicts in the future.

The primary difference between our executive and EPIC's is that ours uses a task-independent procedure to learn task-specific executive process knowledge needed for performing the dual-task combination. Unlike the task-specific EPIC executive rules which, by definition, much be written for each dual-task combination, our procedure should be applicable to any task combination. We consider this a minimalist approach because only the knowledge that is absolutely needed—the knowledge for resolving and avoiding jams—is learned. The third objective of this phase was to investigate the adequacy of our ideas of the nature of the executive process.

The fourth objective was a pragmatic one. By building a performance model, we would be able to see the executive process knowledge that needs to be learned. (This would be done using the definition that any knowledge distinct from that needed to perform the individual tasks is part of the executive process.) We may then be able to cluster the knowledge into classes. Having identified several knowledge classes, we might be able to devise learning procedures for acquiring the knowledge in these classes.

In short, by building a performance model we could first identify the knowledge that is needed for a specific task, then reverse engineer learning procedures for acquiring the knowledge in the hope that the learning procedures are be generally applicable for realizing performance improvements in other tasks.

Though this approach seems analogous to data-fitting (producing learning procedures to learn the desired knowledge), it is hoped that the knowledge classes and the learning procedures or the framework will be sufficiently general to apply to a large class of tasks, while making confirmable and/or plausible predictions about human dual-task acquisition and performance.

6.1 Expert models of the individual tasks

We first built expert models of the individual tasks. This can be justified because in dual-task studies, it is not uncommon for subjects to be trained to some criterion level on the individual tasks before they enter the dual-task condition. See Figure 2.2. These expert models capture the end result of this training. It is important to emphasize that like those studies, this work is only concerned with the development of dual-task skill and not with the prerequisite development of the individual task skills.

As is natural in Soar, the individual tasks are represented as operators with each operator consisting of several steps. (operator implementation rules). See Figure 6.1. The steps for the tracking task are the similar to those used in the EPIC model. Both are motor steps. The first, watch-cur-

Tracking Task	Choice Task
watch-cursor	recognize-stimulus
track-target	verify-stimulus
	send-response

Figure 6.1 Operator descriptions for the tracking and choice tasks

sor, does as its name implies; it moves the eye on the cursor when the visual features that are relevant to tracking are degraded. The second, `track-target`, generates a joystick movement (a 'ply' command) when 1) the distance between the cursor and the target is not `SMALL`, as defined in the EPIC task environment; 2) the manual motor processor is available; 3) the ocular processor is not executing an eye movement command.

The choice-task model assumes that no eye movements are needed; i.e. the subject is fixating on the location where the stimulus will appear, as is typical in CRT tasks experiments. The first two steps for the choice task were incorporated from earlier Soar cognitive modeling research on covert visual attention (Wiesmeyer, 1992). They are cognitive steps and the last is a motor step. The `recognize-stimulus` step is used to determine if the choice stimulus is a left-arrow or a right-arrow. The `verify-stimulus` step confirms that the identified stimulus is relevant to the task. The `send-response` step sends a motor command to press the appropriate key in response to the stimulus.

6.2 Strategies for dual-task behavior

Having created the individual task models, we must now create the first dual-task model. There are two dual-task strategies that can be used for performing two tasks simultaneously.

6.2.1 Task lockout

The first strategy is to have the tasks run in a *lockout* fashion; see Figure 6.2. In the Wickens task, a lockout strategy would produce this behavior: do tracking; when the stimulus has occurred, stop tracking and switch to the choice task; after the stimulus has been responded to, resume tracking. The lockout strategy has the benefit that is it simple to implement and requires little executive process knowledge. It also eliminates the potential for motor conflicts where two different tasks may try to cause the eye to look in different locations at the same time.

6.2.2 Task interleaving

While the lockout strategy has many attractive advantages, it does preclude the possibility of highly efficient, overlapped performance. A second strategy called *interleaving* allows the concurrent execution of the tasks. This is illustrated in Figure 6.3. In the Wickens task, the interleaving strategy would produce behavior somewhat like this: do tracking; when the stimulus has occurred, start doing the choice task while continuing to do the tracking task; when the stimulus has been responded to, the choice task ends and tracking remains active. This strategy provides the *opportunity* for both tasks to operate simultaneously.[1] The behavior seen the EPIC model (Figure 5.2) was due to an interleaved strategy.

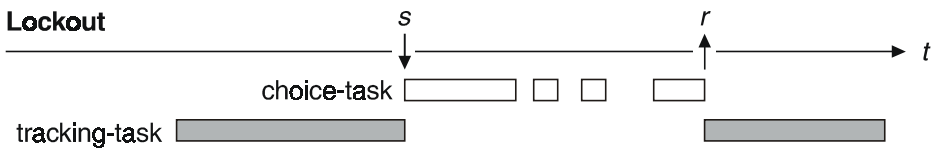


Figure 6.2 The lockout strategy for performing two tasks simultaneously

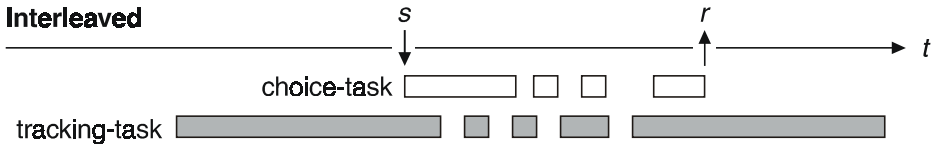


Figure 6.3 The interleaved strategy for performing two tasks concurrently

Concurrent performance does not come without cost as there is now the opportunity for motor conflicts. For example, it is possible for the tracking task to try to move the joystick at the same time the choice task sends a response command. (Recall that although these actions are performed with different hands, EPIC posits a *single* manual motor processor for the control of both hands.) While interleaving promises higher, more efficient performance than the lockout strategy, it may does require a potentially sophisticated executive process to deal with motor conflicts and task priorities.

It is interesting to note that the interleaving strategy is really still a lockout strategy, but at a finer grain size. The grain size for the lockout strategy is a whole task. In contrast, the grain size for the interleaving strategy is the production rule; specifically, the rule that produces a motor command that might conflict with another task..

It is possible to apply these two strategies to the different ways in which people perform tasks as they are learning. When a subject first begins learning a dual-task combination, it is readily observed that their behavior is slow, inefficient, and guarded, in the sense that they try to complete whole tasks at a time. This is very similar to the lockout strategy. By the end of training, the subjects are no longer slow, nor guarded in their behavior. They have developed efficient strategies for coping with performing the two tasks concurrently. Their behavior is as would be expected from the use of an interleaving dual-task strategy. Consequently, a major contribution of practice on dual-task performance may involve enabling a shift from lockout scheduling to fully interleaved scheduling (Meyer & Kieras, 1997a).

One way to think about the differences between these two strategies is in terms of the selection of the task operators. In the lockout strategy, the tracking-task operator is first selected and remains so until the choice stimulus appears. At this point, the tracking-task operator is deselected and the choice-task operator is selected. When the choice-task operator has completed, it is deselected and the tracking-task operator is selected. In the lockout strategy, the operators are selected in a mutually-exclusive manner.

In contrast, the interleaving strategy would allow both operators to be concurrently selected. The tracking-task operator, being the primary task, is permanently selected. The choice-task, can be implemented as a concurrent operator that is also permanently selected. It could also be implemented as a concurrent operator that is selected only when needed (e.g., when the choice stimulus appears) and is deselected when finished (e.g., after the choice-task operator has sent the response). The important point here is that at some time, both operators are active. While the behavior of these two options is essentially equal, we use the former approach.

6.3 Implementation of the executive process

In the work presented in this chapter and those to follow, the executive process will be implemented as an operator called `executive`. All executive process knowledge necessary for the

Wickens task will be part of this operator. This operator will always be selected and will run concurrently with the task operators.[2]

6.4 Creating an expert model in EPIC-Soar

The remainder of this chapter discusses the iterative development of a dual-task performance model, producing a lineage of four models that show a transition from novice behavior to expert behavior for the Wickens task. This development used a generate-and-test approach, where a model is created, tested, and its performance evaluated. If the performance did not match the expert performance, a minimal amount of knowledge (production rules) was *manually* added. This addition of knowledge to the model constitutes a new model. The new model was again tested and evaluated. This cycle repeated until an expert model was found.

Two heuristics are used for adding knowledge. First, we developed the models in the manner that people appear to improve with practice. Initially, subjects generally exhibit very slow, guarded, sequential performance and over time exhibit a fast, efficient concurrent or interleaved performance strategy. Hence, the early dual-task models used a lockout strategy while the later models used an interleaved strategy (Meyer, *et al.* 1995).

The second heuristic applies architectural optimizations to each high-level strategy. These optimizations are anticipatory motor programming and command pipelining Therefore, while using the lockout strategy, we try to one or both of these optimizations. We then switch to the interleaving strategy, and again try to apply one or both of these optimizations.

A final guide in developing these models was to examine the executive process knowledge of the EPIC model and add the knowledge that was deemed appropriate for the model's evolution.

Before discussing the development of the dual-task models, we must say a few words about data collection and performance evaluation. The EPIC architecture, which included a data logging and analysis component, provided the data presented here. Our models were run for 300 trials on each of the twelve conditions (vertical separations). On each of the 300 trials within a condition, the reaction time to the choice stimulus was recorded. The reported RT is the average of these value. RMS tracking error was sampled every 125ms during the two-second period from the stimulus onset to one second after stimulus offset; sixteen samples recorded. With each sample, the running average of the error was updated. The RMS error reported for each condition is the average of all the sampled errors; i.e. number-of-trial * 16. The performance of each model was evaluated qualitatively and quantitatively on how well it matched the observed data.

6.4.1 The basic lockout strategy

To build the first dual-task model, we loaded both individual task models into EPIC-Soar. We then needed to add some knowledge to the system to implement the basic lockout strategy. To implement this strategy in Soar, we manually added two rules to the model: one rule prefers the tracking task when the stimulus is absent; the other rule prefers the choice task when the stimulus is present.

Additionally, the choice task, as defined in Figure 6.1, assumes that the eye is already fixated at the location where the stimulus will appear. In the dual-task situation, where the primary task also uses the eye, this assumption is no longer valid. An additional task facilitation rule, `fixate-on-stim` was needed to move the eye to the choice stimulus when this was necessary. A movement is necessary only when the eye is located such that the detailed perceptual information of the stimulus is unavailable; i.e. the stimulus is outside the fovea.

A single-trial trace of the model is presented in Figure 6.4. This trace, as is true for all EPIC-Soar traces presented in the main chapters of this thesis, is abridged such that extraneous cognitive cycles (called *decision cycles* in Soar) have been removed. (For the cycles that are shown, semanti-

```
52   selected operator: tracking-task & executive
57   command: MOVE PSYCHOBJ21 generated-by tracking-task
60   command: PERFORM PLY generated-by tracking-task
69   command: MOVE PSYCHOBJ21 generated-by tracking-task
75   command: PERFORM PLY generated-by tracking-task
75   command: MOVE PSYCHOBJ21 generated-by tracking-task
81   command: PERFORM PLY generated-by tracking-task
82   command: MOVE PSYCHOBJ21 generated-by tracking-task
89   command: PERFORM PLY generated-by tracking-task
93   command: PERFORM PLY generated-by tracking-task
97   command: PERFORM PLY generated-by tracking-task
101  command: PERFORM PLY generated-by tracking-task
105  command: PERFORM PLY generated-by tracking-task
109  choice stimulus has onset
110  command: FIXATE PSYCHOBJ23 generated-by executive
111  selected operator: choice-task & executive
121  recognized choice-arrow
122  verified left-choice-arrow
123  command: PERFORM PUNCH generated-by choice-task
124  selected operator: tracking-task & executive
125  command: MOVE PSYCHOBJ21 generated-by tracking-task
     finished trial # 2/2
```

Figure 6.4 Trace of the dual-task model using the basic lockout strategy

cally equivalent textual substitutions have been made in the interest of general comprehension. For example, when an operator is selected, a message clearly stating this fact is shown instead of the usual Soar trace notation.) The trace shows this to be a basic lockout strategy. At cycle 52, the tracking-task operator has been selected and performed. At cycle 109, the choice stimulus appears and the eye is moved to it in the following cycle. At cycle 111, the tracking-task operator is terminated and the choice-task operator is selected. When the choice-task completes, the operator is terminated and the tracking task operator is reinstated. We ran this model and collected the data. The average RT and average RMS tracking error results are plotted in Figure 6.6 and are labeled *Lockout (novice)*.

6.4.2 The lockout strategy with preparation

It was no surprise that the basic lockout strategy was entirely inadequate for matching the observed performance. While the average RT results are close, the average RMS error is very bad. It is clear that knowledge needed to be added to the dual-task model to improve its performance.

An improvement was suggested by EPIC architecture. It is possible to improve the average RT match of this model to the EPIC or empirical data by adding knowledge that allows the model to anticipate and prepare for the appearance of the stimulus; i.e. add an anticipatory motor programming rule. Recall from the description of EPIC that when a command is sent to EPIC, it passes through the motor processor in two phases, preparation then execution, both of which take time to perform. However, the total elapsed time of a command can be reduced if, at an earlier time, the command has been prepared for. As was seen in Chapter 5, the model can prepare the eye to look at the location where the choice stimulus will appear before the stimulus appears. Thus, when the choice stimulus appears, the rule that moves the eye on the stimulus will be performed in a shorter time.

```
     selected operator: tracking-task & executive
81   command: PERFORM PLY generated-by tracking-task
90   command: PERFORM PLY generated-by tracking-task
90   command: MOVE PSYCHOBJ26 generated-by tracking-task
94   command: PREPARE PSYCHOBJ25 generated-by executive
96   command: PERFORM PLY generated-by tracking-task
96   command: MOVE PSYCHOBJ26 generated-by tracking-task
102  command: PREPARE PSYCHOBJ25 generated-by executive
105  command: PERFORM PLY generated-by tracking-task
105  command: MOVE PSYCHOBJ26 generated-by tracking-task
109  command: PREPARE PSYCHOBJ25 generated-by executive
111  command: PERFORM PLY generated-by tracking-task
113  choice stimulus has onset
114  command: FIXATE PSYCHOBJ28 generated-by executive
     selected operator: choice-task & executive
124  recognized choice-arrow
125  verified left-choice-arrow
126  command: PERFORM PUNCH generated-by choice-task
128  command: PREPARE PSYCHOBJ25 generated-by executive
     selected operator: tracking-task & executive
129  command: MOVE PSYCHOBJ26 generated-by tracking-task
     finished trial # 2/2
```

Figure 6.5 Trace of the dual-task model using the lockout strategy with preparation

We manually added a single production, `prepare-for-stim`, to generate a preparation for eye movement throughout the tracking task. This iteration of the model was rerun (see the trace in Figure 6.5) and data collected. These results are plotted as *Lockout + prepare* in Figure 6.6. Here we see that preparing for the stimulus improves the average RT match. The tracking error match remains very poor.

6.4.3 Evaluation of the lockout strategy

The poor prediction of the tracking error is the most glaring problem with this lockout strategy. There is a straightforward explanation why this strategy is inadequate. In the Wickens task, the tracking error is measured only from the appearance of the choice stimulus until one second after the choice stimulus disappears. Since the lockout strategy prevents tracking during the choice task, the error rate should be high since the cursor will wander farther and farther away from the target .

It was clear that more knowledge was needed. However, there are no more architecturally-motivated additions *within this strategy*. Since the tracking match problems were due to the lockout strategy, the interleaving dual-task strategy was adopted.

This strategy seems most consistent with the instructions given to subjects as reported by Martin-Emerson & Wickens (1992), "Subjects were instructed to execute a response within the one-second stimulus display period and *to divide their attention equally between the two tasks* [italics mine]." As seen in Chapter 5, this is also the strategy Kieras & Meyer (1995a) found necessary to adopt in their model.

6.4.4 The basic interleaved strategy

Recall that the interleaved strategy can be implemented by concurrently selecting both operators. This is easily accomplished in Soar by using the operator composition technique used in earlier

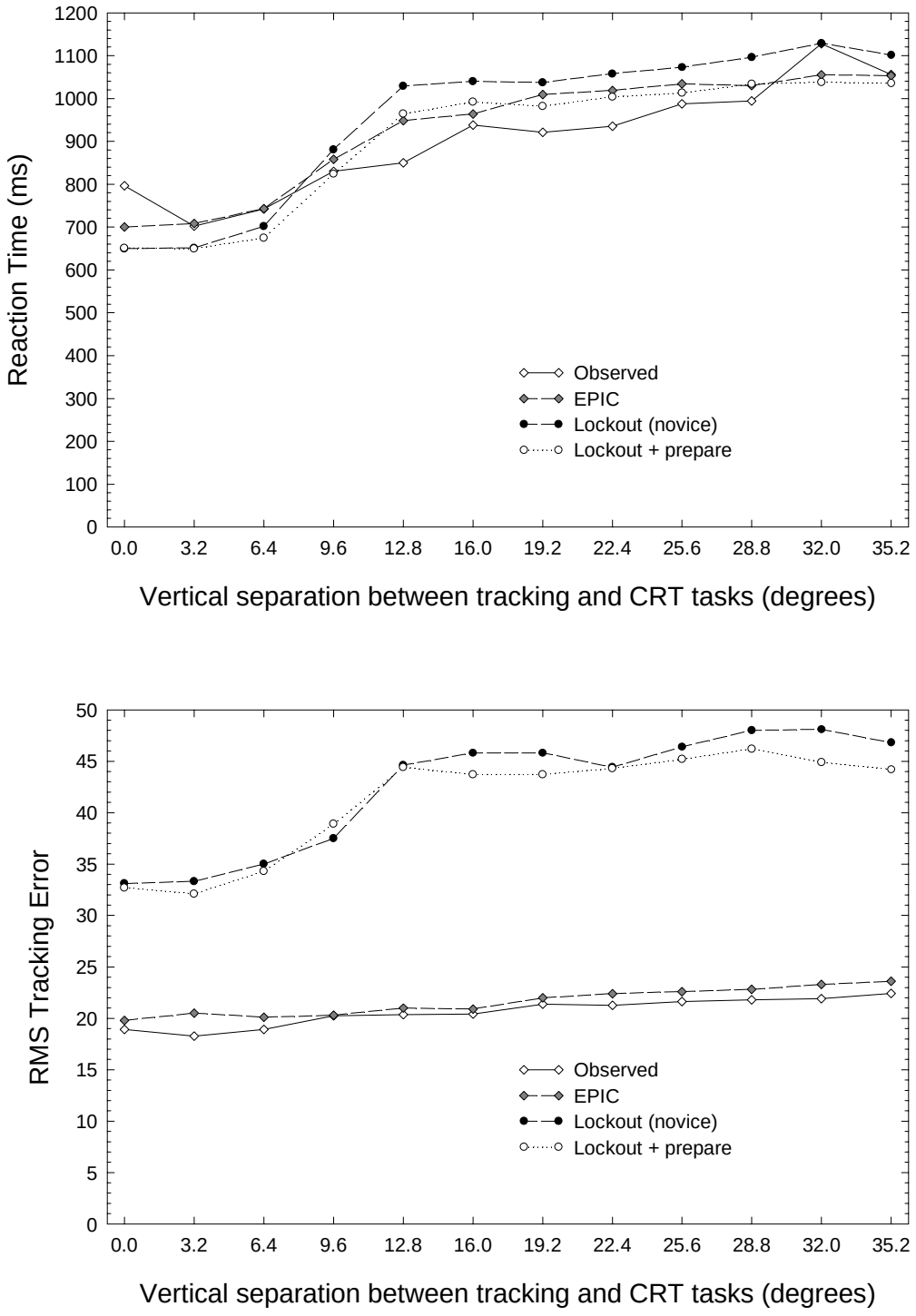


Figure 6.6 The EPIC-Soar dual-task model using the lockout strategy

```
     selected operator: tracking-task & choice-task & executive
57   command: PREPARE PSYCHOBJ30 generated-by executive
57   command: PERFORM PLY generated-by tracking-task
65   command: PERFORM PLY generated-by tracking-task
66   command: MOVE PSYCHOBJ31 generated-by tracking-task
70   command: PREPARE PSYCHOBJ30 generated-by executive
73   command: PERFORM PLY generated-by tracking-task
77   command: PERFORM PLY generated-by tracking-task
79   command: MOVE PSYCHOBJ31 generated-by tracking-task
81   command: PREPARE PSYCHOBJ30 generated-by executive
83   command: PERFORM PLY generated-by tracking-task
87   command: PERFORM PLY generated-by tracking-task
91   command: PERFORM PLY generated-by tracking-task
95   command: PERFORM PLY generated-by tracking-task
99   command: PERFORM PLY generated-by tracking-task
103  choice stimulus has onset
103  command: PERFORM PLY generated-by tracking-task
103  command: FIXATE PSYCHOBJ33 generated-by executive
108  command: MOVE PSYCHOBJ31 generated-by tracking-task
114  recognized choice-arrow
114  command: PERFORM PLY generated-by tracking-task
115  verified left-choice-arrow
115  command: MOVE PSYCHOBJ31 generated-by tracking-task
116  command: PERFORM PUNCH generated-by choice-task
118  command: PREPARE PSYCHOBJ30 generated-by executive
127  command: PERFORM PLY generated-by tracking-task
     finished trial # 2/2
```

Figure 6.7 Trace of the dual-task model using the basic interleaved strategy

Soar work by Covrigaru (1992) and as discussed in Note 2. With this change alone, both tasks now run concurrently, allowing task steps to apply as they may.

There is one pitfall when two tasks are allowed to run interleaved. There is the risk of two or more motor commands being simultaneously sent to the same modality. For example, in the Wickens task, the model may attempt to respond to the choice stimulus and at the same time attempt to move the joystick, both of which use the manual motor system. In native EPIC, this condition is called a "jam". EPIC's reaction is to ignore both commands. The EPIC model of the previous chapter avoided jams because its task-specific, explicit executive process disabled tasks when it was anticipated that a motor conflict might occur.

For reasons stated earlier, we chose not to implement the executive process from the EPIC model in the Soar model. Rather, we continued with the minimalist approach to adding knowledge to the system. First, we allowed both tasks to run concurrently, with the steps of the operators applying as they may. To handle the jams that will arise, we created a task-independent recovery learning procedure. Using declaratively-represented, task-specific preference knowledge (i.e. "prefer the choice-task over the tracking-task"), it decides which command should be sent and then *learns* task-specific rules based on this decision. In the future, when this same situation arises where a jam might occur, the learned knowledge will intervene, obviating the jam and the need for another time-consuming recovery process. (A complete description of this process can be found in Appendix A.) With this approach, the model will incrementally learn executive process rules about

```
     selected operator: tracking-task & choice-task & executive
62   command: PREPARE PSYCHOBJ39 generated-by executive
72   command: PERFORM PLY generated-by tracking-task
76   command: PERFORM PLY generated-by tracking-task
78   command: MOVE PSYCHOBJ40 generated-by tracking-task
82   command: PREPARE PSYCHOBJ39 generated-by executive
84   command: PERFORM PLY generated-by tracking-task
89   command: PERFORM PLY generated-by tracking-task
93   command: PERFORM PLY generated-by tracking-task
97   command: PERFORM PLY generated-by tracking-task
101  command: PERFORM PLY generated-by tracking-task
105  command: PERFORM PLY generated-by tracking-task
109  command: PERFORM PLY generated-by tracking-task
113  command: PERFORM PLY generated-by tracking-task
117  command: PERFORM PLY generated-by tracking-task
121  command: PERFORM PLY generated-by tracking-task
125  command: PERFORM PLY generated-by tracking-task
129  choice stimulus has onset
129  command: PERFORM PLY generated-by tracking-task
129  command: FIXATE PSYCHOBJ42 generated-by executive
134  command: MOVE PSYCHOBJ40 generated-by tracking-task
140  recognized choice-arrow
140  command: PERFORM PLY generated-by tracking-task
141  verified left-choice-arrow
142  command: PERFORM PUNCH generated-by choice-task
142  command: MOVE PSYCHOBJ40 generated-by tracking-task
144  command: PREPARE PSYCHOBJ39 generated-by executive
146  command: PERFORM PLY generated-by executive
     finished trial # 2/2
```

Figure 6.8 Trace of the dual-task model using the interleaved strategy with track-asap

how to avoid jams. This is probably the weakest possible executive process, but there was no guarantee it would be sufficient to produce matching expert performance.

In our model, this recovery procedure was able to build all the jam avoidance knowledge necessary for this task. The key aspect of this learning is that it transforms the general declarative statement into specific procedural knowledge that applies at the exact point where it is needed, i.e. avoiding a jam.

This new model was run. In the trace (Figure 6.7), cycle 56 shows that both operators are selected. Additionally, note a tracking task command being generated at cycles 108, 114, and 115, during the choice-task. The collected data is shown in Figure 6.9 labeled as *Interleaved*.

This model has two qualitative effects: tracking error has been significantly lowered compared to *Lockout + prepare*; tracking error is, as a whole, *independent* of the vertical separation condition; the plot is horizontal. The model is still a poor predictor of tracking error, although it is significantly better than the earlier lockout models.

6.4.5 The interleaved strategy with pipelining

Again, more knowledge was needed to improve performance. We then turned to the EPIC model for ideas of what could be added. There, we observed that the EPIC model used an executive pro-

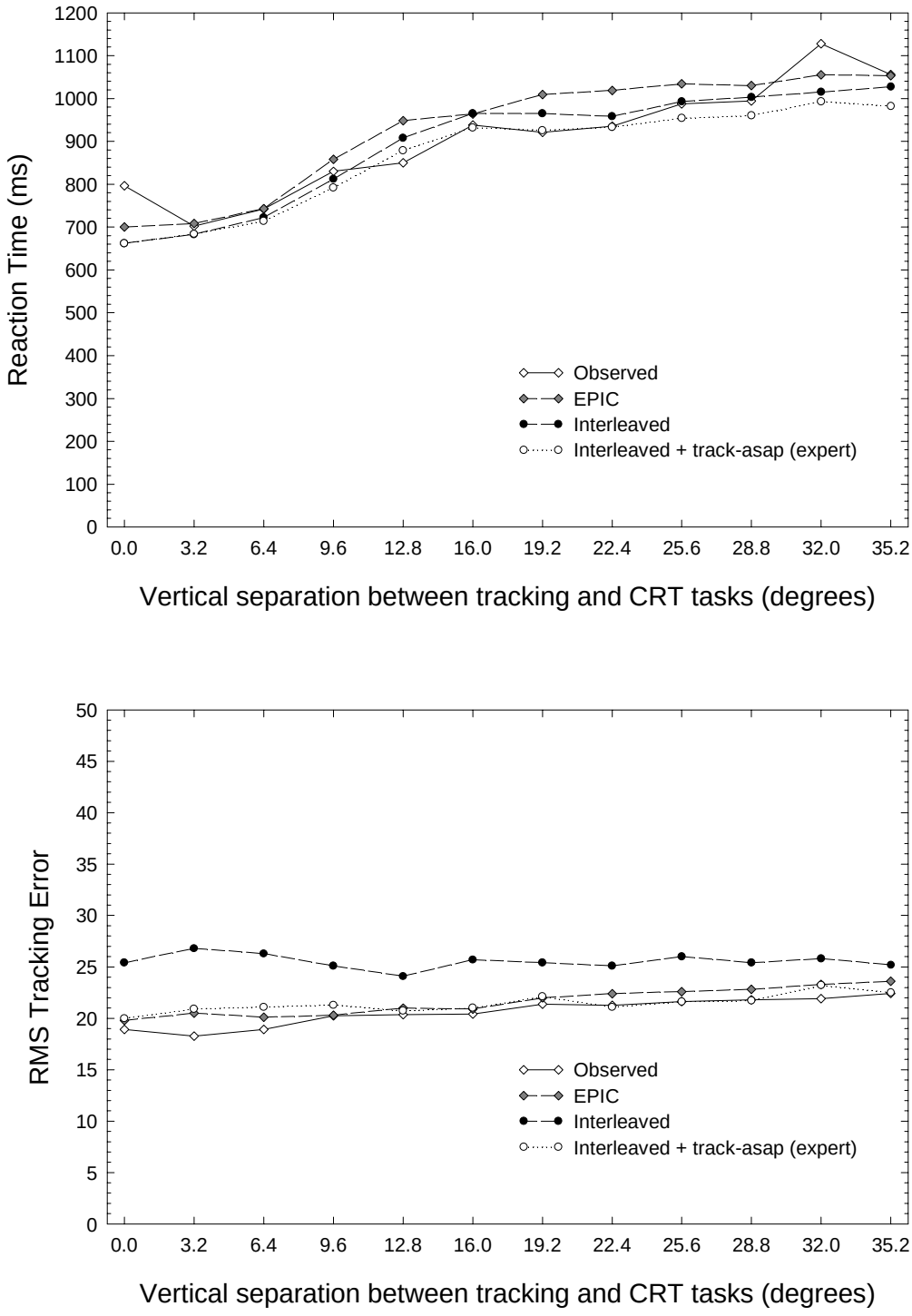


Figure 6.9 The EPIC-Soar dual-task model using the interleaved strategy

duction called `move-cursor-asap` (henceforth, it will be referred to as a `track-asap` rule). This pipelining rule was seen and discussed at length earlier in Chapter 5.

A `track-asap` rule was added and the EPIC-Soar model rerun. As seen in Figure 6.8, the choice task response is sent at cycle 142, with the first joystick command (`track-asap`) executing at cycle 146; approximately 200ms delay. Contrast this with Figure 6.7 where the choice task is responded to at cycle 116 and the first joystick occurring at cycle 124; approximately 400ms.

Figure 6.9 shows the new results labeled as *Interleaved + track-asap (expert)*. The average RT and average RMS error are satisfactory matches to the observed data and the EPIC model; RMS error between the model and the observed data was 62.82 and 1.18, respectively.

6.5 Classifying executive process knowledge

Figure 6.10 depicts the incremental development of the dual-task models just discussed as a lineage of five models, beginning with expert individual task models, and ending with the final dual-task model. The changes/additions that were made at each increment is shown and these represent the executive process knowledge that was needed for our model of the task.

This knowledge can be partitioned into two sets indicating when the knowledge was acquired; either before or after the first dual-task trial (or before/after Stage 2 as depicted in Figure 2.2) was performed. Knowledge that existed before the first trial is called *pre-trial*, while the knowledge that resulted from performance on the task is called *learned*.

We are forced to use our intuition to determine to which class a piece of knowledge belongs since it is difficult to know if a subject possesses a piece of knowledge and when he/she acquired it. However, our intuition can be guided by a heuristic: if it is not reasonable that a piece of knowledge is learned for *every unique dual-task combination*, then the knowledge will can be put in the pre-trial class. Otherwise, it will be considered as learned.[3] A corollary to this heuristic is: if a piece of knowledge is deemed to be task-independent, then it is in the pre-trial knowledge class. If it is deemed to be task-specific, then it is in the learned knowledge class. (This is only an approximation however. The final categorization along the dimension of task-specificty will be presented in Chapter 11 and will show the exceptions to this corollary.)

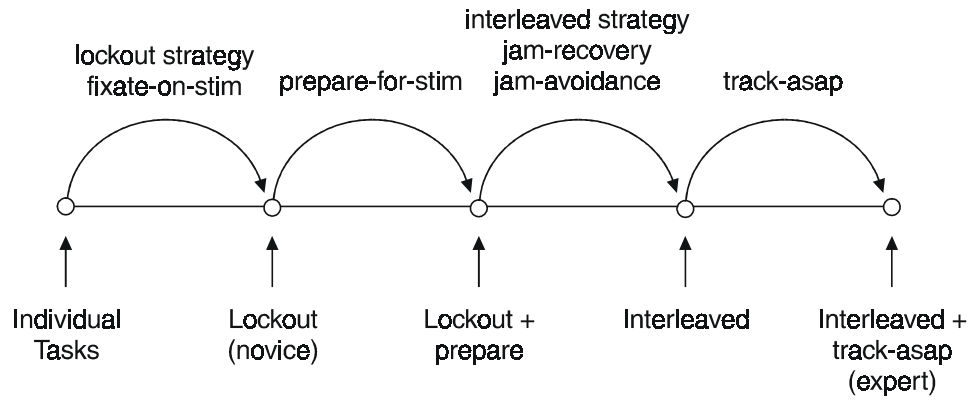


Figure 6.10 The lineage of EPIC-Soar dual-task models

6.5.1 Pre-trial knowledge

Three of the six knowledge additions shown in Figure 6.10 are instances of the pre-trial knowledge class. The first example is the implementation of the lockout dual-task strategy. This is a task-independent strategy. The virtue of task-independence is precisely that the knowledge can be applied across many different tasks. Therefore, by definition, this knowledge is not acquired for every unique dual-task combination, hence, it is considered to be pre-trial knowledge. The second instance is the implementation of the interleaved strategy. This instance is considered to be pre-trial knowledge for the same reasons as the lockout strategy.

The third instance is the jam-recovery learning procedure that recovers from motor conflicts. This is a task-independent procedure, and once again must be considered pre-trial knowledge.[4]

6.5.2 Learned knowledge

It is worthwhile to re-emphasize that the research objective of this thesis is to: Develop a computational, task-independent framework for modeling the acquisition of the knowledge that subjects acquire through practice on a selected dual-task combination. Hence, the knowledge in the 'learned' knowledge class is *exactly* what should be acquirable with the framework we seek.

The remaining additions of Figure 6.10 are instances of the learned knowledge class; knowledge that is acquired from experience in performing the dual-task combination. These rules are `fixate-on-stim`, `prepare-for-stim` and `track-asap`.

The `fixate-on-stim` rule moves the eye to the choice-task stimulus when its features are degraded. This rule might be considered a task-specific instantiation of a general (or primitive) rule (say, `perform-saccade`), which moves the eye to some desired object when its features are degraded. Although this is a task-specific rule, and according to the heuristic should be considered as 'learned' knowledge, a case can be made for this being an pre-trial rule. For example, it is conceivable that this instantiation of the general rule is created after the task instructions ("look at the choice stimulus when it appears") are given to the subject. This kind of instruction explicitly states to look at the stimulus, giving the subject enough information with which to create the rule before dual-task trials begin, making this an pre-trial rule.

On the other hand, it is equally conceivable that the instructions were not that explicit. If the subject did not "read between the lines" to infer the need for such a rule, then when the stimulus appeared during the first trial, the subject at that point would have to create (possibly by means-ends analysis) and use this rule. As both arguments appear to be plausible, we will assume that the `fixate-on-stim` rule is pre-trained knowledge.[5]

The `prepare-for-stim` and `track-asap` rules are both task-specific rules and are instances of anticipatory motor programming and command pipelining, respectively. As was seen for the `fixate-on-stim` rule, arguments can be made for these two rules being pre-trial knowledge. However, they tend to be much less persuasive as there is a key difference between these rules. The `fixate-on-stim` rule is *fundamental* to producing dual-task performance. Without saccading to the choice-stimulus when it appears, the subject would simply not be able to perform the choice-task and would not produce dual-task performance. In contrast, the `prepare-for-stim` and `track-asap` rules are not at all necessary for dual-task performance, and in fact represent optimizations of the task.

Knowing that subjects are given to producing very poor first-trial dual-task performance, it is more likely that they are more concerned with complying with the task instructions (by using `fixate-on-stim`) rather than trying to produce the optimized performance that the `prepare-for-stim` and `track-asap` rules afford. Therefore, these optimized rules should be learned during experience on the task. It is not obvious however what learning procedure/s could be used to acquire this knowledge.

An additional instance of the learned knowledge class is the jam-avoidance rules that are created by the jam-recovery learning procedure. These rules are task-specific and are clearly learned during dual-task trials in this model.

6.6 Discussion

In this chapter, we presented a lineage of four dual-task models that was used to identify the additional knowledge required to perform two tasks interleaved at an expert level for the Wickens task. Each model in the lineage represents the addition of knowledge to the previous model. Much of the knowledge has been manually added, much of it is essentially Soar analogues of the EPIC productions, which on its own is uninteresting. However, the larger goal is to have a system that learns the rules that humans learn during dual-task practice.

The merit of this phase of the work then is that it takes the first small steps towards that goal: (1) an expert performance model has been constructed in EPIC-Soar; (2) the model produced a good match to the observed data; (3) our use of a minimalist executive controller appeared to be sufficient *for this task*; (4) the knowledge needed to progress from novice to expert (executive process knowledge) has been identified; (5) two general knowledge classes have also been identified: pre-trial and learned; (6) we can now focus on learning procedures that might be used to acquire the knowledge in the learned class; (7) one such task-independent acquisition procedure (the jam-recovery procedure) which learns how to deal with some of the initial problems of interleaved performance has already been demonstrated.

The following chapter presents the development of the framework for acquiring the other two instances—anticipatory motor programming and pipelined rules—of the learned knowledge.

Notes to Chapter 6

[1] The task-interleaving strategy only provides the *opportunity* for concurrent performance. If the task stimuli are spatially co-located (in the case of visual stimuli) or the different stimuli use different perceptual modalities (visual and auditory for example), then concurrent performance is possible. However if visual stimuli are spatially separated by a large distance, or the effector or task device are shared between the tasks, or if the task devices are spatially separated by a large distance, then concurrent performance may not be possible.

[2] It is commonly said that Soar can only pursue on operator at a time and hence imposes an inherent cognitive bottleneck. Strictly speaking, this is not true. When an operator is selected, the architecture creates a *single* data structure for the selected operator. This structure appears in working memory as a working memory element such as:

```
(<o> ^name tracking-task)
```

This structure contains a `^name` attribute which holds the name of the operator. All rules that are associated with this operator (e.g., the rules which implement the tracking task) will test if this working memory element is present.

A feature of Soar is that an attribute of a working memory element can be multi-valued; i.e., a single attribute can have several values. With this feature, the operator structure can be elaborated with multiple operator names:

```
(<o> ^name tracking-task ^name executive)
```

As a result of this, both operators are enabled; i.e., all the rules that match on the `tracking-task` operator are able to fire and all the rules that match the `executive` operator are also able to fire. Because any number of operators can be performed in parallel using this technique, there is no *inherent* cognitive bottleneck in Soar.

[3] In Chong & Laird (1997), the knowledge classes were reported as: task, innate, experiential, and strategy. Since that report, a clearer partitioning of the executive process knowledge came to light. This reformulation will be presented here.

[4] Recall from Chapter 2 that executive process knowledge can be defined as the knowledge necessary to produce dual-task performance that is distinct from the knowledge used to perform the constituent tasks. Most of the work in modeling executive processes has focused its the role in promoting expert dual-task *performance*. This work, however, is concerned with dual-task acquisition *and* performance. By our definition of executive process, we must consider *learning procedures* to be part of the executive process knowledge since they are used as the model transitions from novice to expert performance.

[5] Earlier (Section 6.1 and Figure 6.1), we mentioned that subjects in real choice-task experiments do not need to move their eyes during trials, therefore `fixate-on-stim` could not be part of the choice-task model. However, at some point in the study (at the beginning of a block of trials, for instance), subjects do indeed need to move their eyes to the point where the stimulus will appear. In hindsight, it is clear that subjects possess this knowledge and that it is performed in service of the choice task. Included `fixate-on-stim` as part of the choice task would have made moot these arguments about the source of the rule.

CHAPTER 7

A FRAMEWORK FOR ACQUIRING EXECUTIVE PROCESS KNOWLEDGE

In the previous chapter, the `prepare-for-stim` and `track-asap` rules were classified as part of the learned knowledge class and it was not clear how these rules could be learned. In this chapter, the development of a task-independent learning procedure for acquiring these rules will be presented. First, a key observation about these rules is presented. A hypothesis about the initial knowledge possessed by subjects will follow. Our observation and hypothesis together suggested a task-independent learning procedure. The procedure will be described and tested on a fictitious task in this chapter but applied to the Wickens task in the next.

7.1 OBSERVATION

The `prepare-for-stim` rule is an example of an anticipatory motor programming rule; it prepares the motor processor for an upcoming command. If the preparation phase of a soon-to-be-

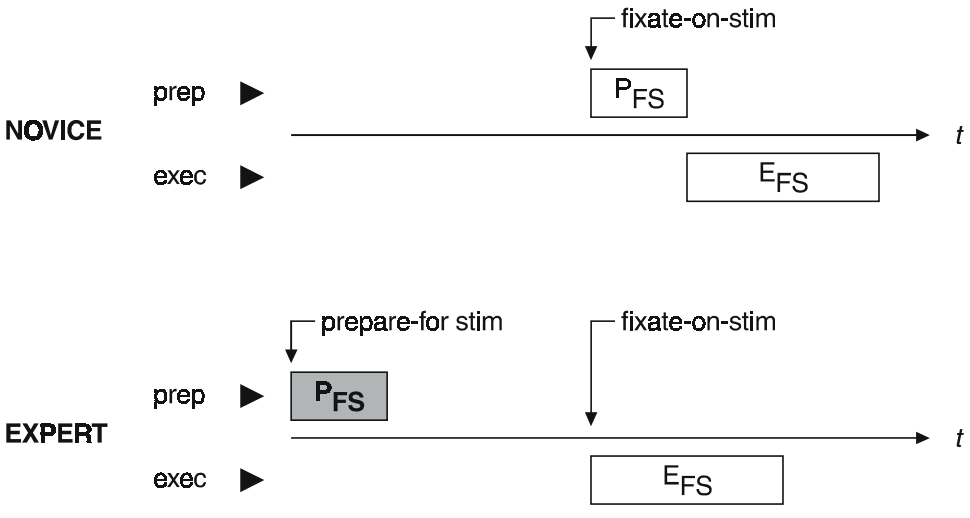


Figure 7.1 Casting anticipatory motor programming as a partial-command promotion

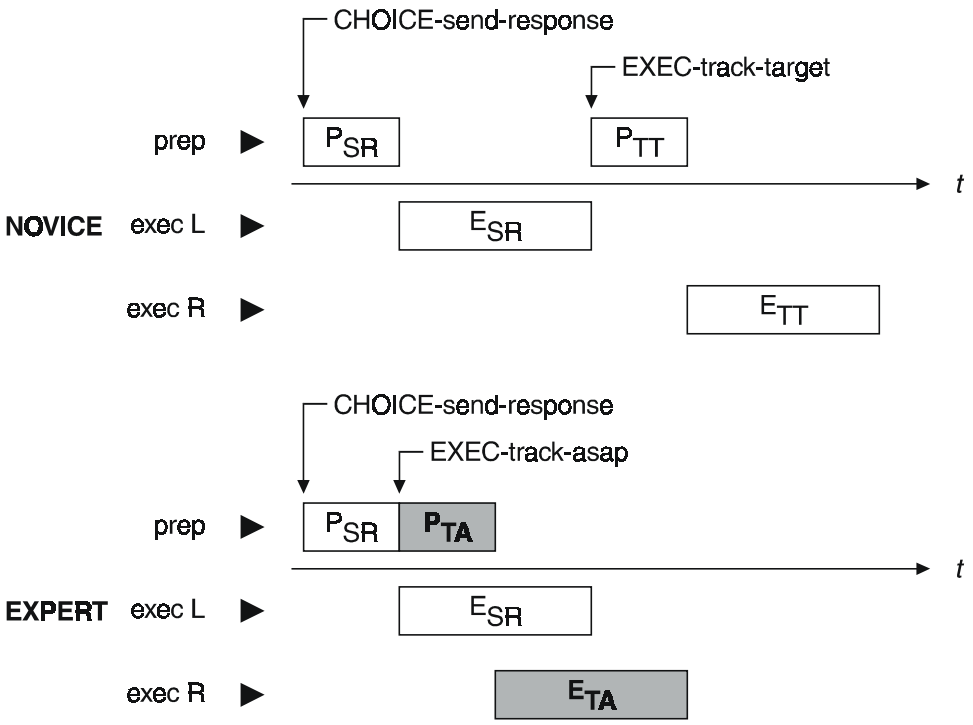


Figure 7.2 Casting pipelining as a whole-command promotion

sent command can be completed ahead of time, then the preparation phase (and the time required to prepare) can be avoided when the command is finally sent, resulting in a significant reduction in the time to execute the command; recall Figure 3.4.

The `track-asap` rule is an example of a command pipelining rule. The EPIC motor processors can execute only one command at a time, but pipelining is a way to make the motor processor process two consecutive commands (for the same modality) at the same time, resulting in a potentially significant reduction in the time to execute the second command; recall Figure 3.5.

When comparing the novice dual-task rules with these expert rules, a key observation was made: the expert rules produce improved performance by executing commands, or parts of commands, *chronologically earlier in the task*, as though they were *chronologically promoted*.

Figure 7.1 demonstrates this for the anticipatory motor programming rule. The novice dual-task model executed `fixate-on-stim` when the choice-task stimulus appeared. By using the `prepare-for-stim` rule, the expert dual-task model 'moved' the preparation of the fixate (P_{FS}) to an earlier time so that when the choice-task stimulus appeared, the fixate action can be immediately executed since the features were prepared ahead of time.

Command pipelining moves the whole motor command to an earlier time such that the command will be issued as soon as the execution of a previous command (that uses the same processor) has begun. This is illustrated in Figure 7.2. The novice dual-task model executed a `track-target` rule at the completion of the execution of `send-response`. In contrast, the expert model used the `track-asap` rule to cause a tracking command to be sent earlier; as soon as the preparation of the `send-response` had been completed.

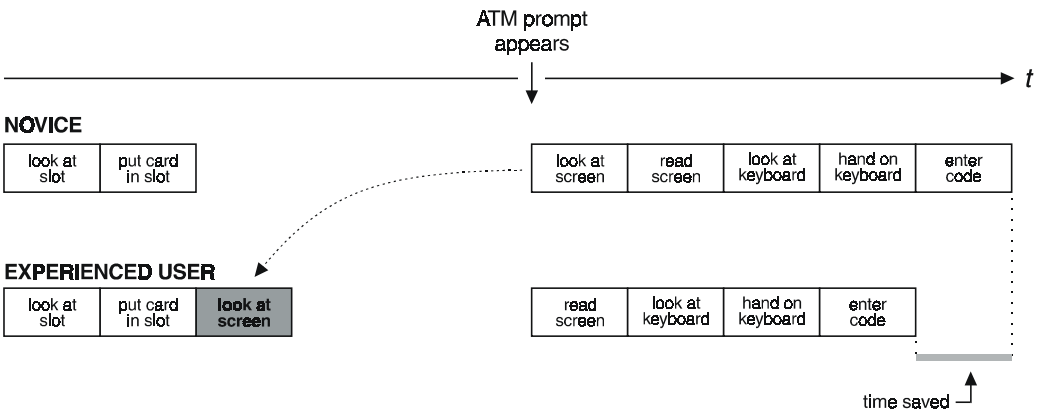


Figure 7.3 Casting movement pre-positioning as a promotion

There is a third task-independent method for improving performance, which was not used in the Wickens task, is called *movement pre-positioning* (Wood, Kieras, & Meyer, 1994; Kieras, Wood & Meyer, 1995a, 1995b). The idea here is that performance can be improved by positioning an effector (a hand or an eye, for example) at a location *before* the action at that location is needed.

Consider the early steps of using an ATM (automated teller machine) where, after the user has put their bank card in the slot, the machine prompts them to enter an access code on the provided keypad, and they enter their code. A strategy for this behavior is depicted by the top diagram in Figure 7.3.

There are improvements that can be made on this strategy however. If we were to observe an experienced user of ATMs, we might notice that after putting their card in the slot, the user would pre-position their eye to the screen in anticipation of reading the prompt. The lower diagram in the figure illustrates this strategy. It shows that one step has been moved to a point earlier in the sequence of behavior. This movement results in a reduction of the time to accomplish the task. This movement is clearly a kind of promotion. The promotion is superficially similar to anticipatory motor programming in that performance gains come about by doing something that is a prerequisite to the action earlier rather than later. In the case of anticipatory motor programming, only the preparation is done earlier, whereas here, the prerequisite command is moved to an earlier event.

7.2 Hypothesis about a subject's initial knowledge

Though the observation that the expert learned rules could be cast as promotions of novice rules was interesting, it provided no further insight toward learning procedure. It became apparent that there may not be enough knowledge in our model for a learning procedure to operate.

In an attempt to add the necessary knowledge, we turned our attention to hypothesizing about the relevant initial knowledge a novice dual-task subject possesses. The intuition is that this initial knowledge plays a significant role in the learning process. This seemed reasonable since many (if not all) learning mechanisms or procedures create new knowledge from previously existing knowledge. The hope was that a) including this knowledge would make our model be a closer approximation to the novice dual-task subject; and b) this knowledge would suggest a learning procedure amenable to the idea of promotions. To this end, we asked and answered a fundamental question:

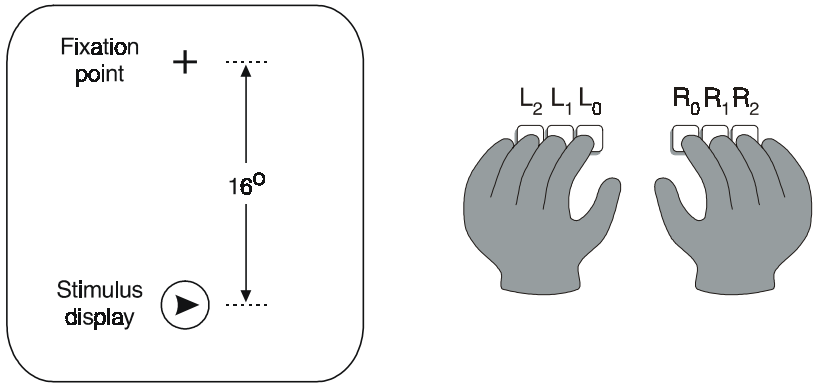


Figure 7.4 The example task, a two-choice reaction-time task

"What initial knowledge does a subject possess *after* receiving task instructions but *before* beginning to perform a task?" Though it is impossible to know exactly what is in a subject's head after reading task instructions, we will make two assumptions: a) that the subject understands the task instructions, and b) is sufficiently motivated to perform the task and abide by the instructions.

7.2.1 Example task: A two-choice reaction time task

To facilitate understanding of the upcoming discussion, we will use the fictitious example task[1], depicted in Figure 7.4. The task description and instructions are:

> When a trial starts, you are to look at the fixation point. A stimulus will appear in the stimulus circle (which is displaced 16 degrees below the fixation point). When the stimulus appears, look at the stimulus. If it is a left arrow, respond by pressing the key sequence `L0, L1, L2, L0` with the index, middle, ring, and index fingers of the left hand; if it is a right arrow, respond by pressing the key sequence `R0, R1, R2, R0` with the index, middle, ring, and index fingers of the right hand. Look back to the fixation point to begin a new trial.

The initial performance model used for this example task is the same as the Wickens task choice task model, with the addition of two ocular command rules: the first to saccade from the fixation point to the stimulus, and a second to saccade from the stimulus back to the fixation point at the end of the trial.

7.2.2 Chronological task strategy data structure

Our answer to the question of what knowledge a subject gains from reading dual-task instructions was that this knowledge is a detailed plan, or strategy, of how to perform the task. This strategy was realized by a data structure called a *chronological task strategy data structure*. This structure represents a subject's knowledge about the chronological ordering of perceptual events, and the motor commands that are required (per the task instructions) at the occurrence of each perceptual event. The structure is also intended to captures a subject's interpretation biases, and pre-performance reasoning about the task. (This latter aspect is discussed in Appendix C.) This structure is related to work on the declarative representation of skill (Anderson, 1981; Bovair & Kieras, 1991).

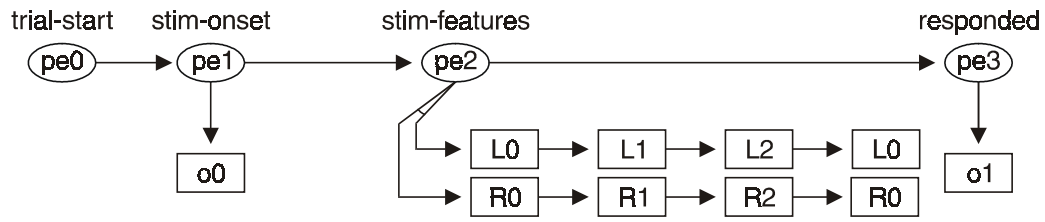


Figure 7.5 A strategy data structure for the example task

The data structure consists of two types of nodes: perceptual event nodes and motor command nodes. Figure 7.5 shows a possible chronological task strategy structure for the two-choice reaction-time task.[2] The structure is read as follows:

> Sometime after the trial starts, the stimulus will appear. When it does, the model performs an ocular command to saccade to the stimulus. At a later time, the stimulus features will arrive in working memory (more on this later) at which time the model will identify the kind of stimulus and will execute one of the prescribed manual motor chains; either the left or right hand key sequence. After the complete response has been made, the model drives the eye back to the fixation point.

The oval nodes are *perceptual event* nodes or *pe-nodes*. The nodes in boxes are *motor command* nodes or *mc-nodes* which contain the motor commands to be executed and can be linked together to form command chains when a sequence of actions *for the same modality* are required, as is the case in this task.

This structure not only represents the chronological ordering of perceptual events and the motor commands as stated, implied, or inferred from the task instructions, but also captures the *preconditions* of motor commands: an arrow from a pe-node to mc-node or from mc-node to mc-node denotes dependency. Therefore, before command `L0` can be generated, the `stim-features` event must have occurred. Similarly, command `L1` can be generated only after command `L0` has been generated.

There are two aspects of this structure that are present solely because this work is situated in a a cognitive system that is constrained by both perception and action. The first is the `stim-features` perceptual event. This event is necessary due to the perceptual processors in EPIC. In the human visual system, when a stimulus appears, the features that define the object are not instantly available. Rather, the features take time to propagate from the sensory system, through perception, and finally into working memory. Additionally, the feature propagation time is dependent on the feature type; i.e. color, shape, location, or size. EPIC models this process, therefore the `stim-features` event is used to signal the arrival of relevant stimulus features. Only after these features have arrived can cognition identify the object and select and produce the appropriate response/s.

The second aspect concerns the links between mc-nodes in the motor chain. When a command is sent to a motor processor, EPIC returns motor status messages that report on the state of each phase (whether each phase is busy or free). Additionally, EPIC provides proprioceptive feedback on the state of the *effector* performing the command; this applies only to manual commands.

In our example task, when the command to press the `L0` key is sent, EPIC accepts the command and reports back when the preparation phase is free, and at a later time when the execution

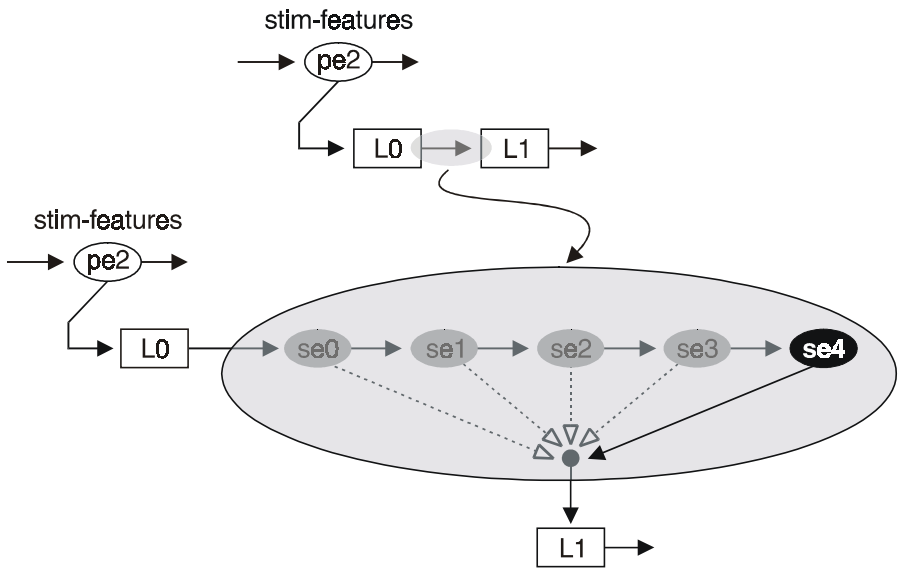


Figure 7.6 Substructure of command chains

phase is free. Because this is a manual command, EPIC's *tactile* sensory processor will report when the key is touched, when the downstroke is completed, and finally, when the upstroke is completed; recall Section 3.3.3.

These messages can be collectively used to create a substructure between chained commands. Figure 7.6 illustrates these messages as a list of *status event nodes* or *se-nodes* and the simple arrow between the `L0` and `L1` expands into a list of se-nodes. (In this thesis, the substructure between chained commands is always shown as just arrows in the interest of graphical clarity. Nevertheless, it is always the case that this substructure exists between consecutive commands in a chain.)

The figure illustrates that after the `L0` command has been sent, the aforementioned messages—represented as the status nodes `se0` = processor free; `se1` = execution free; `se2` = key-touched; `se3` = key-depressed; `se4` = key-released—are returned from EPIC.[3] The dotted arrows between the event nodes and the subsequent command, `L1`, represent potential dependencies, while the solid arrow shows the current dependency; `L1` is preconditioned on `se4`, `key-released`.

This substructure as a whole can be characterized as defining a *dependency* or *precondition space*. This space allows a model to produce a potentially wide range of chained performance styles, from *cautious* to *aggressive*, or in more relevant terms, from *novice* to *expert*. In this figure, the dependency of `L1` on `se4` produces a cautious, novice performance style because it requires that the `L0` key be released before the command for the second keypress, `L1`, could be sent. An aggressive, expert performance style could be realized if `L1` was dependent on `se0`, while intermediate performance styles would be generated by dependencies on `se1` through `se3`.

7.3 The promotion-learning procedure

As was stated earlier, the observation that chronological promotions can produce performance improvement was interesting but did not inspire a learning procedure. With the subsequent invention of the chronological task strategy data structure, it became clear that *principled* chronological

promotions could be created from this structure. A straightforward task-independent *promotion-learning* procedure for acquiring the learned knowledge was devised and will now be described.

The learning procedure performs three styles of promotions: *prepare promotions*, *event promotions*, and *chain promotions*. Before describing each style, a brief overall description of how the learning procedure performs these promotions will be presented.

7.3.1 Overview

To use this learning procedure, the user must provide a chronological task strategy data structure (represented declaratively in working memory) and a procedural novice model which performs the task exactly as it is represented in the structure. The model is run. At some point, a command rule, say `rule-A`, will match, fire, and generate a motor command, say `command-A`. In response, task-independent *promotion suggestion rules* may fire. These suggestion rules look at both the generated command and the strategy data structure to determine which promotion, if any, can be legally executed (as defined by the promotion guidelines found in Appendix C). Assuming a promotion suggestion rule does fire, the suggested promotion style is invoked and a promoted command rule, `rule-pA`, is learned. With the exception of the rule due to a prepare promotion, the newly learned rule produces the same motor command (the action side of the rule is the same as the original rule), but it is preconditioned on a chronologically earlier event (the condition side of the rule is different from the original rule).

At the same time that the promoted rule is learned, the strategy data structure is updated to reflect this promotion. This updating also results in rules being learned. These strategy structure rules are used to regenerate the promoted structure when the EPIC-Soar system is restarted. It is of *paramount* importance that the strategy structure representative of the procedural knowledge for performing the task since the promotion procedure uses the structure to decide what future promotions are possible.

The new command rule is immediately available for use in task performance. When this new rule eventually fires, it too will be evaluated by the promotion suggestion rules. The learning procedure as implemented runs concurrently with task performance, so performance is not at all hindered by the application of the procedure. Learning is incremental and produces potentially gradual improvements in performance as a result. We now describe the three promotion styles.

7.3.2 Prepare promotion

The *prepare promotion* style creates anticipatory motor programming rules and is defined as:

> When a motor command that is preconditioned on pe-node `pe(t)` is executed, a prepare-promotion suggestion is generated *if the command satisfies the prepare promotion guidelines* (see Appendix C). The prepare promotion style applies and a new *prepare* rule is learned. This new rule is preconditioned on pe-node `pe(t-1)` and produces a `(prepare <action>)` command where `<action>` is the command produced by the command on pe-node `pe(t)`. The strategy structure is modified to reflect this promotion.

Figure 7.7 demonstrates the application of this promotion style to the strategy structure of the example task. The black boxes with white text highlight the changes to the structure. They indicate that two prepare promotions for ocular commands were performed. The first, `po0`, was generated for the saccade which responds to the appearance of the stimulus, `o0`. Because of this promotion, the reaction time would be improved by the same amount of time needed to prepare the saccade features. The second `po1` was generated for the saccade that returns the eye to the fixation point. This prepare does not affect the reaction time in this task.

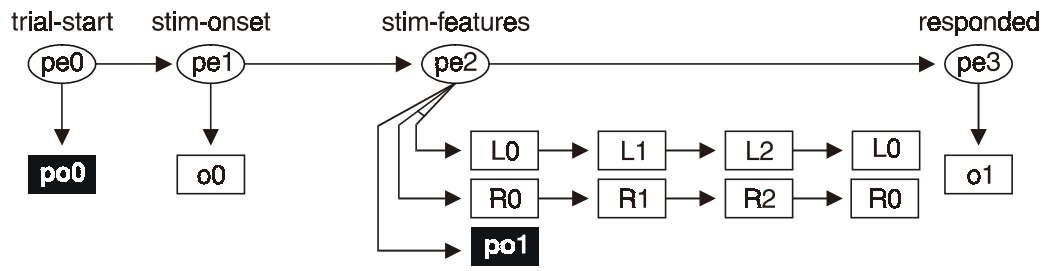


Figure 7.7 The strategy data structure after all *prepare promotions*

7.3.3 Event promotion

The *event promotion* style creates movement pre-positioning rules and is defined as:

> When a motor command that is preconditioned on pe-node `pe(t)` is executed, an event-promotion suggestion is generated *if the command satisfies the event promotion guidelines* (see Appendix C). The event promotion style applies resulting in a promoted command rule which produces the same command but is now preconditioned on pe-node `pe(t-1)`. The strategy structure is modified to reflect this promotion.

Figure 7.8 demonstrates the change of the structure of Figure 7.7 due to this promotion style. The figure shows only a the prepare rule was a candidate for event promotion. The resulting rule causes the preparation of the `o1` saccade, `po1`, to occur on the `stim-onset` event rather than the `stim-features` event.

Note that this promotion resulted in an ocular motor chain consisting of `o0` and `po1`. This demonstrates one of the guidelines for performing event promotions: if the `pe(t-1)` node already has a modality-specific chain (where a chain can be of size = 1), then any event-promoted command from `pe(t)` that uses the same modality must be appended to the end of this chain.

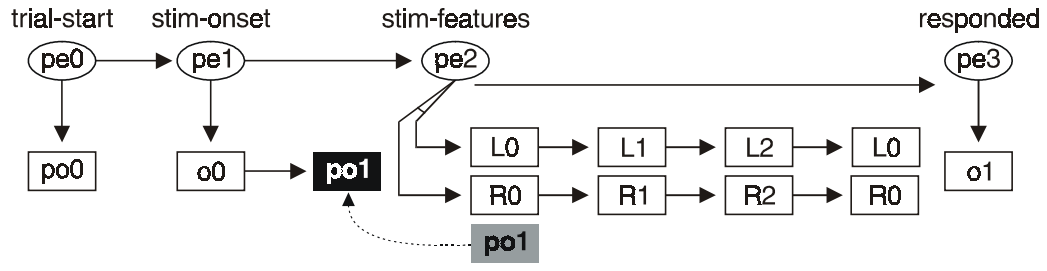


Figure 7.8 The strategy data structure after all (one) *event promotion*

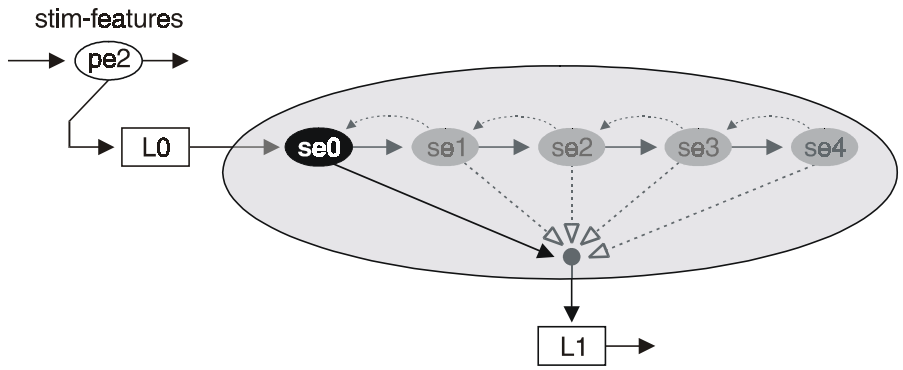


Figure 7.9 Chain promotion of `L1`

7.3.4 Chain promotion

Chain promotion uses the substructure shown in Figure 7.6 to allow chained motor commands to be preconditioned on chronologically earlier motor processor status events; i.e. chain promotions change the performance style from cautious to aggressive. This promotion is defined as:

> When motor command that is preconditioned on se-node `se(t)` is executed, a chain-promotion suggestion is generated *if the command satisfies the chain promotion guidelines* (see Appendix C). The chain promotion style applies resulting in a promoted command rule which produces the same command but is now preconditioned on s-node `se(t-1)`. The strategy structure is modified to reflect this promotion.

Chain promotions directly create command pipelining rules. To illustrate this, consider the promotions of the `L1` command as shown in Figure 7.6. The first time this command is generated, a chain promotion suggestion will be generated. This promotion style will create a new command rule which is preconditioned on `se3`, `key depressed`, instead of `se4`, `key released`. When this newly learned `L1` rule fires at a later time, it will again be chain promoted, causing it to be preconditioned on `se2`, `key touched`. This behavior continues with every application of the promoted `L1` command rule until it is preconditioned by `s0`, `processor free`, , at which time all chain promotions for `L1` will be exhausted (see Figure 7.9). The original rule would generate the `L1` command after the `L0` key had been released. In contrast, this new rule generates the `L1` command as soon as the feature preparation phase is completed for `L0`. This is clearly a pipelining style of rule.

Chain promotions will occur for all the other chained commands though only the of `L1` has been described. In effect, chain promotions reduce the time between consecutive commands in a chain, this effect is represented in the Figure 7.10 by the shortened arrows between chained motor nodes.

Command pipelining rules are also created, though indirectly, by both the prepare promotions and event promotions discussed previously. This is true because pipelining rules can only be created within motor chains both prepare promotions and event promotions can create motor chains. An example of this is illustrated by the movement of the `po1` node from `stim-features` to

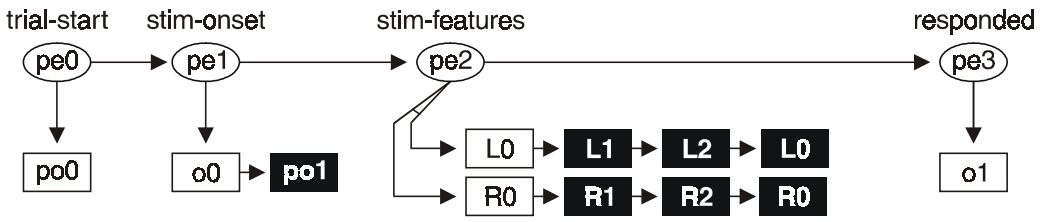


Figure 7.10 The strategy data structure after all *chain promotions*

`stim-onset` (as shown in Figure 7.8) in which an ocular motor chain was created. The `po1` rule was then pipelined, as seen in Figure 7.10.

The procedural knowledge represented by the structure in Figure 7.10 is the final expert model for this task. The next section will present the results of applying the procedure to our example task.

7.4 Promotion-learning applied to the example task

The strategy structure of Figure 7.5 was hand-coded as declarative knowledge and given to EPIC-Soar. Also, a procedural task performance model that was congruent with the behavior represented by the strategy structure was provided to EPIC-Soar.

Figure 7.11 shows the performance data gathered over three phases. This data were collected over two task conditions: "2-CRT task" and "1-CRT task". The "2-CRT task" condition is the two-choice reaction task as described here. The large (~100ms) fluctuations during the "Novice" and "Expert" phases was at first quite puzzling. A colleague suggested that maybe these fluctuations were related to responding to different stimuli. To explore this idea, the simulated task environment was modified to always produce the same stimulus; say the left arrow. New data were collected and are presented in Figure 7.11 under the label *1-CRT task* although the same performance model and strategy structure were used as in the 2-CRT task condition.[4]

The difference between these plots is explained by a yet unmentioned aspect of feature preparation in the EPIC motor processors. For this discussion, we will refer to keypress commands which are of the form `(PUNCH <hand> <finger>)`. These commands require that two features be generated; the hand and finger.

When a command (say the manual motor command for `L0` in this task) is sent to a motor processors, movement features are first created and the action is executed. The command that presses `L0` is `(PUNCH LEFT INDEX)`. Unstated thus far is the fact that the processor saves the movement features. When a subsequent command is sent, a comparison is done between the previous and current commands. If the commands are the same, then the stored features are simply recalled, and the action is immediately executed resulting in a feature preparation time of 0ms.

If the new command is to press `L1`, which is defined as `(PUNCH LEFT MIDDLE)`, the comparison finds that the <hand> specification is the same, but the `<finger>` specification is different. Feature preparation then consists of reusing the hand feature and creating the correct finger feature. Feature preparation time will be 50ms nominally.

If, however, the next command is to press `R1`, `(PUNCH RIGHT INDEX)`, then the comparison finds the `<hand>` specification to be different. This requires that both features be generated even though the `<finger>` specification is the same. In this case, feature preparation costs 100ms nominally.

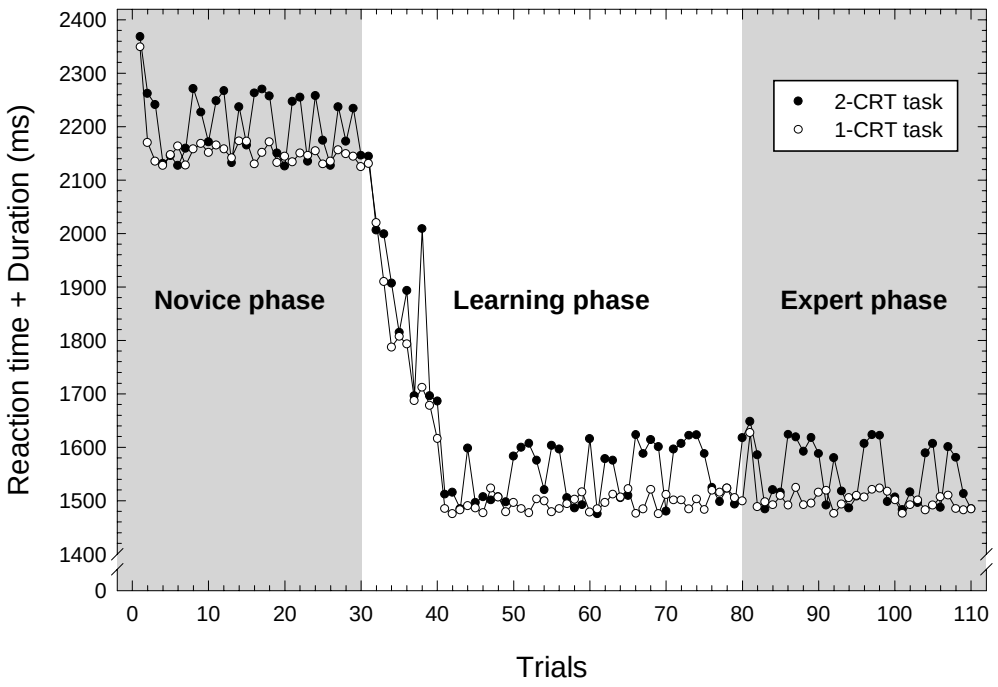


Figure 7.11 Transitioning from novice to expert performance on the example task

From this explanation, the ~100ms fluctuations in the 2-CRT trace that were initially perplexing are easily attributed to the full feature preparation—the preparation of both `<hand>` and `<finger>` features—that is needed when the stimulus in the current trial differs from the last.

We now describe the three phases. The "Novice" and "Learning" phases occurred during an 80-trial run. In the "Novice phase", the novice task model was run with the promotion learning procedure disabled and spanned the first 30 trials. This phase was done to get a clear baseline of novice level performance. Notice the large (~200ms) improvement between the first and second trial. This is not a learning effect, but rather due to first-time feature preparation. If this baseline phase was not first run, then this first-time feature preparation time would have been included in "Learning phase", giving the illusion of a much bigger learning effect.

The model was programmed to pause at the end of the 30th trial of the 80-trial run. At this time, the flag `(<s> ^enable-promotions t)` was added to working memory by means of a single production. The run was resumed, beginning the "Learning phase" which spanned trial 31 to 80. The learning is found to be gradual, requiring approximately 10 trials to transition from the novice performance level of approximately 2150ms down to the expert level of 1500ms; a 650ms improvement. The last 40 trials of the "Learning phase" are flat and stable showing that all learning has been exhausted.

At the end of the 50-trial learning phase, EPIC-Soar had learned 75 rules. Forty-five of the 75 rules were strategy modification rules which were created as the structure was modified due to promotions. These rules capture the evolution of the strategy data structure.

The remaining thirty rules were motor command rules. However, only eight of the thirty motor rules were applicable: one for `po0`, one for `po1`, and one for each keypress at position two or greater in the motor chains. The 22 other motor rules were no longer applicable because when a motor command is promoted, the rule that *initially* produced the command (the command which

NOVICE BEHAVIOR

```
606  add event:choice-stimulus-onset
606  command:FIXATE PSYCHOBJ82 generated-by choice-task
616  recognized choice-arrow
617  verified left-choice-arrow
617  add event:choice-stimulus-features
618  command:PERFORM PUNCH LEFT INDEX generated-by choice-task
626  command:PERFORM PUNCH LEFT MIDDLE generated-by choice-task
635  command:PERFORM PUNCH LEFT RING generated-by choice-task
644  command:PERFORM PUNCH LEFT INDEX generated-by choice-task
645  add event:responded-to-choice-stimulus
646  command:FIXATE PSYCHOBJ74 generated-by choice-task
     finished trial # 8/80
```

EXPERT BEHAVIOR

```
82   command:PREPARE PSYCHOBJ155 generated-by executive
107  add event:choice-stimulus-onset
107  command:FIXATE PSYCHOBJ158 generated-by choice-task
109  command:PREPARE PSYCHOBJ156 generated-by executive
116  recognized choice-arrow
117  verified left-choice-arrow
117  add event:choice-stimulus-features
118  command:PERFORM PUNCH LEFT INDEX generated-by executive
120  command:PERFORM PUNCH LEFT MIDDLE generated-by executive
124  command:PERFORM PUNCH LEFT RING generated-by executive
129  command:PERFORM PUNCH LEFT INDEX generated-by executive
130  add event:responded-to-choice-stimulus
131  command:FIXATE PSYCHOBJ156 generated-by choice-task
     finished trial # 2/30
```

Figure 7.12 Traces of a novice trial and and expert trial for the 1-CRT condition

caused a promotions suggestion to be made) is disabled by the new, earlier-firing rule. Only the newest rules for a give motor command are ever applicable.

To verify that this learned knowledge would produce performance at the same level at the end of the "Learning phase", the EPIC-Soar system was restarted and reloaded as normal and was also given the 75 learned rules. The model was re-run for 30 trials (shown as trials 81-110 in the figure). All 45 of the structure modification rules fired first, reproducing the evolution of the initial strategy structure due to promotion learning. See Figure D.5 in Appendix D. After these firings, the structure corresponded to Figure 7.9, which is the behavior implicit in the eight applicable motor command rules. The data collected from this run were shown in the "Expert phase" of the figure and demonstrates that performance is as expected.

Figure 7.12 shows abridged traces of a single trial for the novice and expert phases. (Traces of the learning phase can be found in Appendix D.) These traces show cognitive cycle number and the model's action during that cycle. They confirm what has been shown in both the strategy structure modification in Figures 7.5-7.9 and the performance plot of Figure 7.11.

The novice trace reveals that a) there are no preparatory commands in the novice model; b) from stimulus onset, it takes ten cycles (500ms) before the stimulus is recognized; c) the inter-key

delay is roughly nine cycles (~450ms); d) from stimulus onset, almost 40 cycles (~2000ms) transpire until the last key in the sequence has been pressed.

The expert trace, in contrast, shows that a) anticipatory motor programming is used; b) from stimulus onset, it takes eight cycles (400ms) before the stimulus is recognized—two cycles (100ms) less than the novice model, resulting from the preparation performed at cycle 82; c) the inter-key delay has been reduced four to five cycles (~200-250ms) due to the pipelining of the key sequences motor chain; d) from stimulus onset, 22 cycles (~1100ms) transpire until the last key in the sequence is been pressed—about half that of the novice model, due to the cycles saved in b) and c).

Notes to Chapter 7

[1] Before selecting this example task, we first searched the literature for a recent study that used a simple dual-task combination consisting of visual-manual tasks *and* reported on both learning and final performance. The difficulty with the very few studies found was that none of them reported on the details of the task that are necessary for modeling. For example, there was no mention of the sizes of the stimuli, or the distance of the subject from the stimuli (or computer screen on which the stimuli was presented). Stimulus size can have an effect on performance since EPIC's visual sensory processor models the varying quality of information produced by the different concentric retinal zones. Though not related to stimulus size, the effect of this difference in information quality will be used to explain the poor performance match of Figure 9.1.

[2] There can be no single correct structure to represent a task since a) it is possible to have many different strategies that all accomplish the same task; b) task strategies may change as task conditions change; and c) biases in interpreting the task instructions can result in different strategies.

[3] Manual commands have five possible preconditions; `se0`, `se1`, `se2`, `se3`, and `se4`. Ocular commands have only two possible preconditions—`se0` and `se1`—since EPIC provides no proprioceptive feedback for ocular commands.

[4] The results of the 1-CRT condition have been used here merely as a diagnostic and should *not* be interpreted as a prediction of behavior on a 2-CRT task where only one kind of stimulus is ever presented.

CHAPTER 8

Applying the acquisition framework to the Wickens task

The last chapter presented the development of an acquisition framework—the chronological task strategy data structure, the promotion learning procedure, and the promotion guidelines found in Appendix C. The framework was applied to a fictitious example task and shown to give expert performance. The current chapter will apply this framework to the Wickens task. The chronological task strategy data structures will be discussed, followed by the application of the learning procedure and the results.

8.1 Strategy structure for the Wickens task

Figure 8.1 shows the strategy task structure for the individual tasks of the Wickens task. Both structures capture the behavior of the models for the individual tasks as shown in Figure 6.1. The motor nodes are: `L` = punch the left button; `R` = punch the right button; `o0` = saccade to the cursor; `m0` = ply the joystick such that the cursor is moved towards the target.

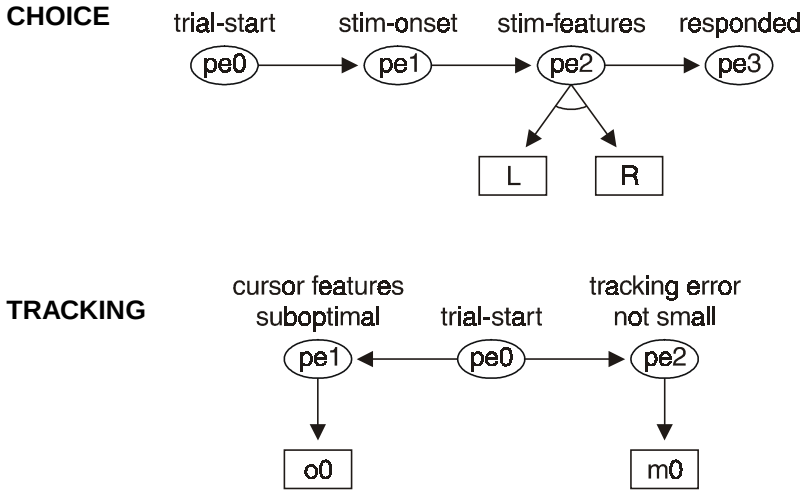


Figure 8.1 Individual task strategy structures for the choice and tracking tasks

Figure 8.2 shows the novice dual-task structure for the Wickens task. It is primarily the union of the individual task structures, with the addition of three task facilitation motor command nodes. The first, `o1`, is an ocular command that moves the eye to the stimulus; the same `fixate-on-stim` rule as was discussed in Chapter 6. This rule facilitates the choice task, and therefore allows dual-task performance. The second, `o2`, moves the eye back to the cursor.[1] This rule is a non-urgent version of the `EXECUTIVE-put-eye-back-on-cursor-asap` rule that was used in the EPIC model (see cycle 55 in Figure 5.2). The command `m1` can be thought of as a command that resumes tracking as soon as a response has been made. It is a non-urgent version of the EPIC rule `TRACKING-TASK-move-cursor-asap` that was used in cycle 71 in Figure 5.2. In the EPIC-Soar model, the motor command rules that generate `o1`, `o2`, and `m1` are not part of either the tracking task or the choice task, and are, by definition, coded as part of the `executive` operator..

8.2 Applying promotion-learning

The strategy structure of Figure 8.2 was hand-coded as declarative knowledge and given to EPIC-Soar. Also, a novice (lockout) dual-task performance model whose behavior was congruent with the strategy structure was provided to EPIC-Soar.

The model was put through three runs, each of which consisted of 100 trials per condition. First, the novice model was run with the promotion learning procedure disabled for the purpose of determining a performance baseline. The results of this run are labeled as *Lockout (novice)* in Figure 8.3.

In the second run, promotion learning was enabled by the same method described in Chapter 7. The model then was put through a short training run of 20 trials on one condition. This allowed all promotions to be created—only ten trials were actually needed for all promotions to be learned. Figure 8.4 shows the effect of the structure as the promotions were applied.[2]

A prepare promotion was applied to the ocular command `o1`. This resulted in the addition of a new motor command node, `po1`. Three event promotions were performed that promoted the nodes `m1` from `pe3` to `pe2`, forming a manual motor chain. Command node `o2` was promoted

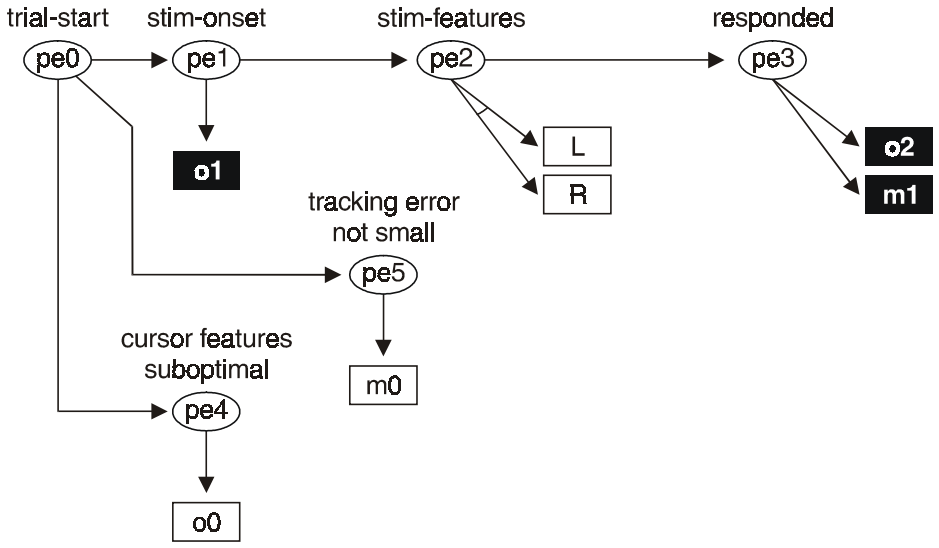


Figure 8.2 Strategy structure for the Wickens task

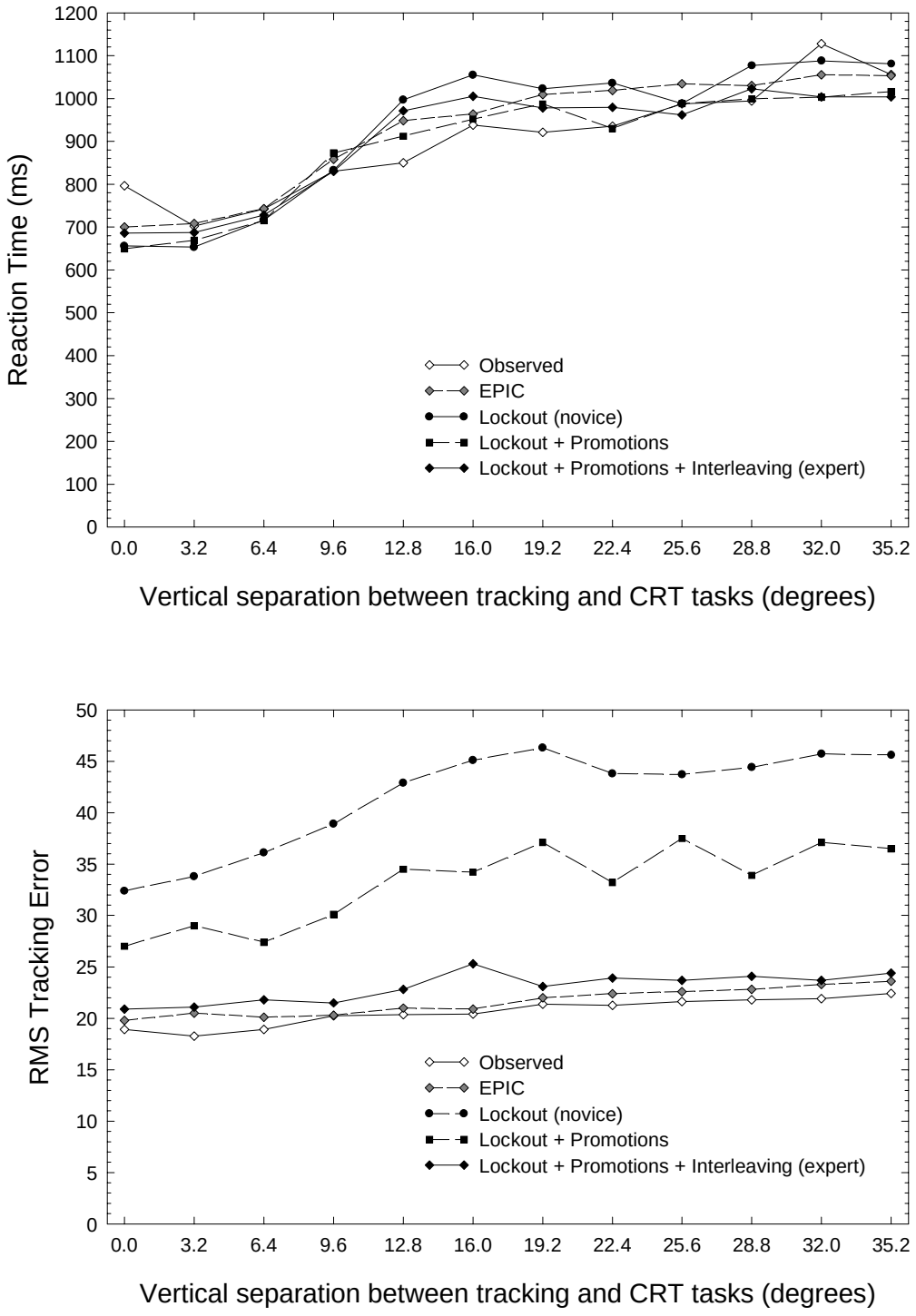


Figure 8.3 Transitioning from novice to expert performance on the Wickens task

from `pe3` to `pe2`, and later was again promoted from `pe2` to `pe1`, producing an ocular motor chain. Finally, several chain promotions were performed to gradually migrate the ocular and manual motor chains from cautious to aggressive behavior. After chain promotion, the `o2` rule behaves exactly like the `EXECUTIVE-put-eye-back-on-cursor-asap` rule used in the EPIC model, and the `m1` rule is a `track-asap` rule.

By the quiescence of promotion learning in the practice run, EPIC-Soar had learned 43 new rules. Twenty-eight of the 43 rules are strategy modification rules which were created as the structure was modified due to promotions. These rules capture the evolution of the strategy data structure.

The remaining fifteen rules are motor command rules. Six of these are jam-avoidance rules generated by the jam-recovery learning procedure. The remaining nine are due to promotions. Of these nine promotion rules, only three are applicable at the end of training (for reasons previously discussed in Chapter 7). They are: one for `po1`, one for `o2` and one for `m1`.

After the training run, the model was reset and run for 100 trials across all conditions. Figure 8.3 shows the results as the trace labeled *Lockout + Promotions*.

Thus far, the model is still using a lockout dual-task strategy. For the final run, the model was switched to use an interleaving dual-task strategy. This switch was manually performed by the addition of a rule that added the element `(<s> ^enable-interleaving t)` to Soar's working memory and allowed both tasks to run concurrently.

Data were gathered on each run and is shown as the trace labeled *Lockout + Promotions + Interleaving (expert)* in Figure 8.3. The figure shows the transition from the baseline novice model to

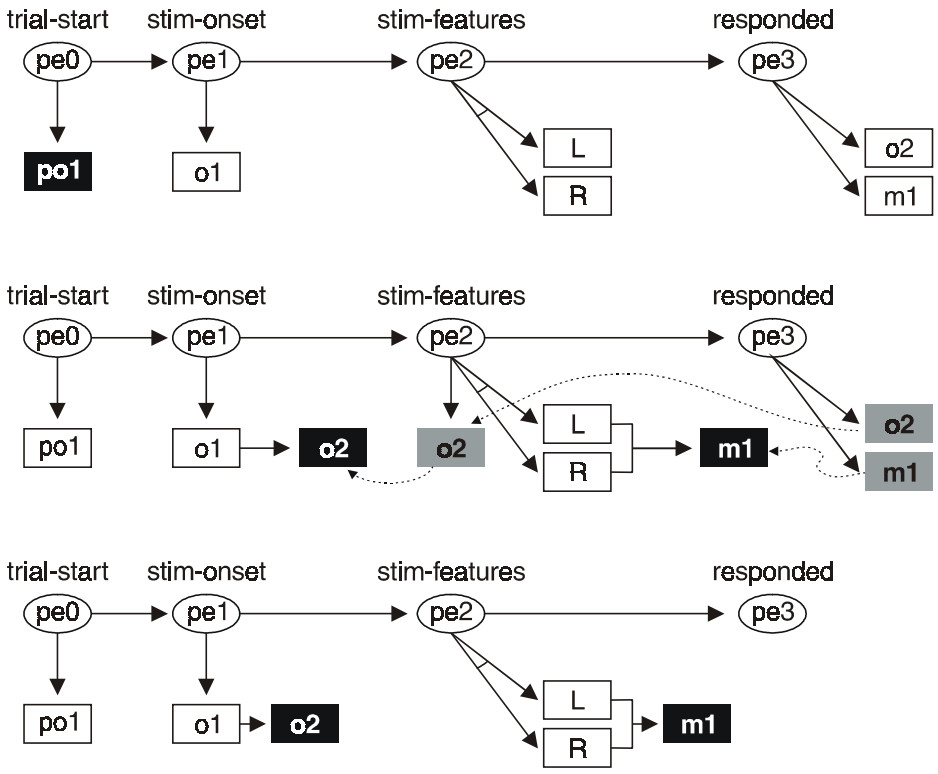


Figure 8.4 The evolution of the strategy structure for the Wickens task

NOVICE BEHAVIOR

```
     selected operator: tracking-task & executive
123  command: PERFORM PLY generated-by executive
133  command: PERFORM PLY generated-by tracking-task
133  command: MOVE PSYCHOBJ252 generated-by tracking-task
138  command: PERFORM PLY generated-by tracking-task
139  command: MOVE PSYCHOBJ252 generated-by tracking-task
142  command: PERFORM PLY generated-by tracking-task
146  command: PERFORM PLY generated-by tracking-task
150  command: PERFORM PLY generated-by tracking-task
154  command: PERFORM PLY generated-by tracking-task
     selected operator: choice-task & executive
156  add event: choice-stimulus-onset
156  command: FIXATE PSYCHOBJ255 generated-by executive
167  recognized choice-arrow
168  verified left-choice-arrow
168  add event: choice-stimulus-features
169  command: PERFORM PUNCH generated-by choice-task
     selected operator: tracking-task & executive
174  add event: responded-to-choice-stimulus
174  command: MOVE PSYCHOBJ252 generated-by executive
177  command: PERFORM PLY generated-by executive
     finished trial # 3/100
```

EXPERT BEHAVIOR

```
     selected operator: tracking-task & choice-task & executive
139  command: PREPARE PSYCHOBJ344 generated-by executive
145  command: PERFORM PLY generated-by tracking-task
150  command: PERFORM PLY generated-by tracking-task
154  command: PERFORM PLY generated-by tracking-task
158  command: PERFORM PLY generated-by tracking-task
162  command: PERFORM PLY generated-by tracking-task
166  command: PERFORM PLY generated-by tracking-task
170  command: PERFORM PLY generated-by tracking-task
173  add event: choice-stimulus-onset
173  command: FIXATE PSYCHOBJ348 generated-by executive
175  command: MOVE PSYCHOBJ345 generated-by executive
180  command: PERFORM PLY generated-by tracking-task
182  recognized choice-arrow
183  verified right-choice-arrow
183  add event: choice-stimulus-features
183  command: MOVE PSYCHOBJ345 generated-by tracking-task
186  command: PERFORM PUNCH generated-by choice-task
190  command: PERFORM PLY generated-by executive
     finished trial # 3/100
```

Figure 8.5 Novice and expert traces for the Wickens task

the post-promotions model, and finally to the expert interleaved model. With the exception of the "bump" at the 16° condition, the final model produced a good overall visual match to the observed RT and tracking data; the RMS error between the model and the observed data was 63.54 and 2.55, respectively.

Figure 8.5 shows abridged traces of a single trial for the novice and expert phases. (The events that trigger the tracking task commands—`watch-cursor` and `track-target`—have been removed to reduce the length of the trace.) They confirm what has been shown in both the strategy structure modification in Figure 8.4 and the performance plot of Figure 8.3. In the expert trace, note the use of the learned anticipatory motor programming rule (cycle 139). Also note the halving of the response-to-ply delay in the expert trace (cycles 186-190) as compared to the novice trace (ccles 169-177). This improvement is due to the learned pipelining rule, `track-asap`.

Notes to Chapter 8

[1] The alert reader will notice that the `o2` rule was not used in the expert model developed in Chapter 6. Frankly, its omission in that model was an oversight. However, in those models, the behavior of this rule was provided by the `watch-cursor` rule in the tracking-task. During the lockout-strategy, this rule could not apply until the tracking-task was restarted. During the interleaved-strategy, the rule would fire sometime after the `fixate-on-stim` rule has fired.; see cycle 108 in Figure 6.7.

[2] The branches of the strategy structure that pertain to the tracking task as shown in Figure 8.2 have been left off this and subsequent strategy structure figures because no promotions could apply to these command nodes based on the guidelines defined in Appendix C.

CHAPTER 9

Predictions of the acquisition framework

The last chapter demonstrated that the acquisition framework is able to produce post-learning behavior that results in a good match to the observed human data. This chapter will present the predictions of the framework in the context of the Wickens task.

9.1 Post-learning performance is dependent on the dual-task strategy

In the runs just discussed, the model used the same assumption as in Chapter 6: novice subjects initially use a lockout dual-task strategy but eventually transition to an interleaved dual-task strategy. Meyer, *et al.*, (1995) provide support for this assumption. This approach will be referred to as the *initial-lockout* model.

Realizing that some subjects may never actually use a lockout strategy but rather might begin with an interleaved dual-task strategy, we decided to re-run the model using this configuration. This will be referred to as the *initial-interleaving* model. Our expectation was that the promotion learning procedure would produce *exactly* the same final performance, making the prediction that the dual-task strategy initially used did not matter to final performance.

To confirm our expectation, the EPIC-Soar system was restarted with interleaving enabled and put through two runs of 100-trials per condition. As before, to get a performance baseline, the first run was made with promotion learning disabled. The results, labeled as *Interleaving (novice)*, of the novice interleaved model are shown in Figure 9.1. The tracking error is markedly better than the novice lockout model. (See Figure 8.3.) This is no surprise since, as noted in Chapter 4, the tracking error is recorded only for two seconds after stimulus onset. Since an interleaved strategy was used, tracking would occur while the choice task was being performed thus producing better tracking performance.

For the second run, promotion learning was enabled. As before, the model was put through a short training run of 20 trials on one condition in order for all promotions to be created—only six trials were needed for all promotions to be learned.

An analysis of the rules revealed that at the completion of promotion learning, EPIC-Soar had learned only 28 new rules; fifteen fewer than before. Eighteen of the 28 rules were strategy modification rules which were created as the structure was modified due to promotions. These rules capture the evolution of the strategy data structure. The remaining ten rules were motor command rules. Four of these were jam-avoidance rules generated by the jam-recovery learning procedure. The remaining six were due to promotions and of these six, only two were applicable at the end of training, for the same reasons as before. They are: one for `po1`, and one for `m1`. It was evident from this analysis that the post-learning behavior might not be as expected.

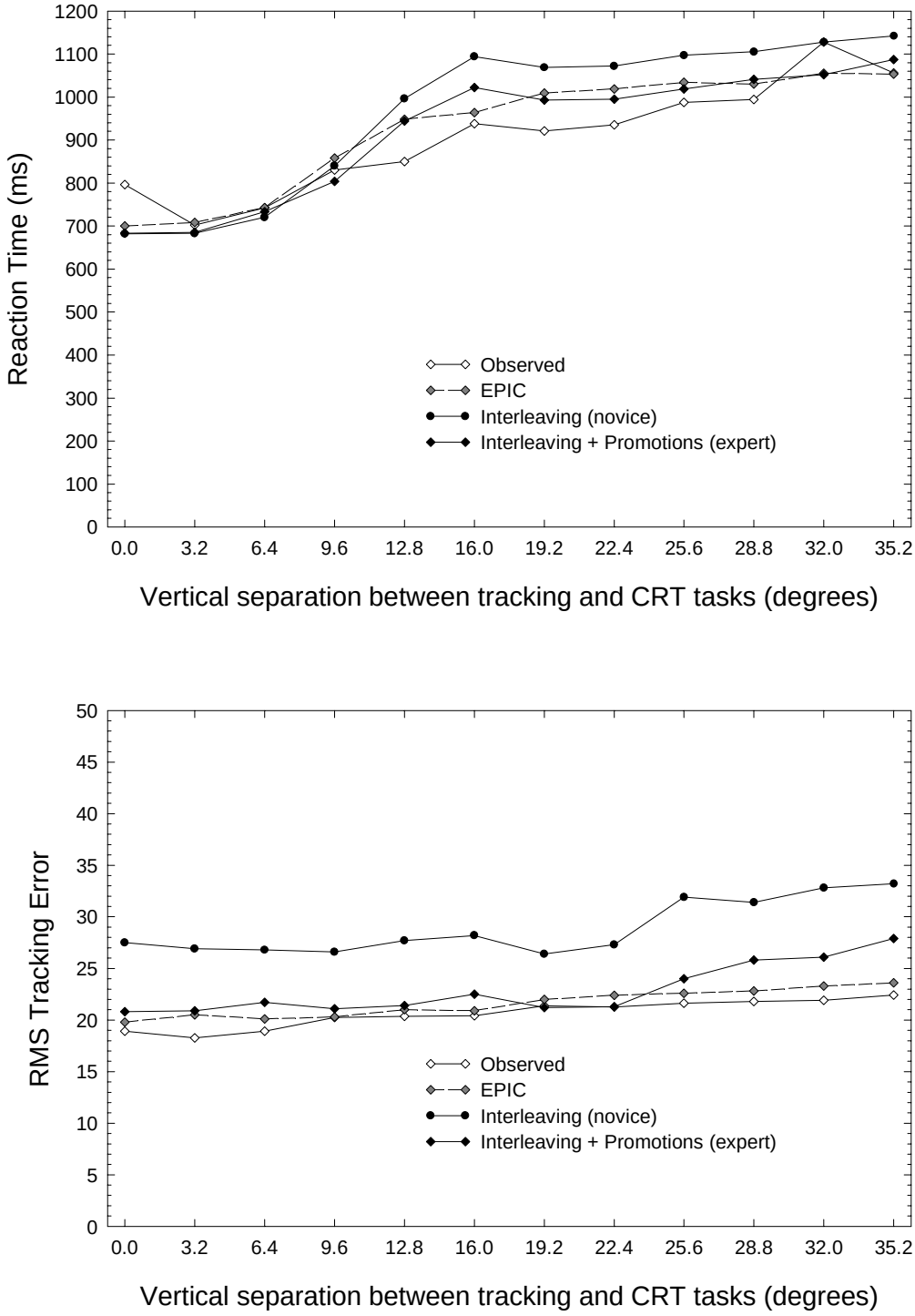


Figure 9.1 Results of learning when initially using an interleaving dual-task strategy

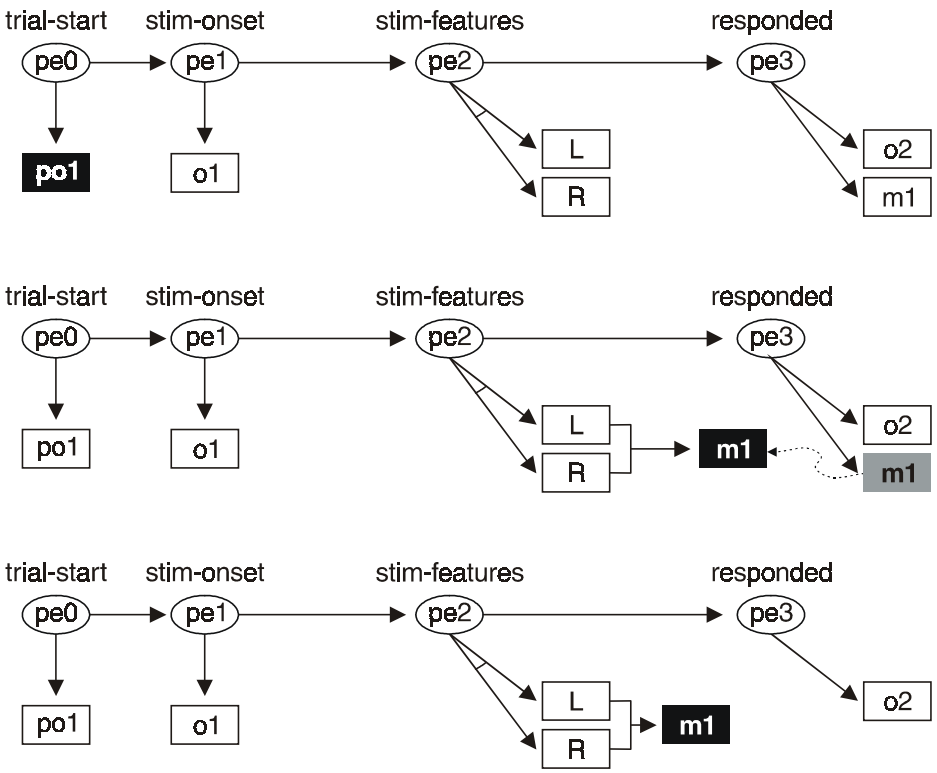


Figure 9.2 The evolution of the strategy structure for the initial-interleaving model

After the 20-trial training run, the model was reset and run for 100 trials across all conditions; see the trace labeled *Interleaving + Promotions (expert)* in Figure 9.1. Contrary to our expectations, we found that the final performance was in fact *not* the same as the *initial-lockout* model (by initial-lockout, we are referring to the model developed in Chapter 8). Specifically, the tracking error performance was worse. The RMS error of the initial-lockout and initial-interleaving models relative to the observed data was 2.55 and 2.79 with a correlation of 0.89 and 0.74 respectively. Unlike that of the initial-lockout model (in which there was a condition-dependence across all conditions; note the slight upward slope), this initial-interleaving model appeared to be *condition independent* (flat) for the first two-thirds of the conditions (0 to 22.4) followed by a pronounced condition-dependence (upward slope) for the last third of the conditions.

In order to understand why the model would make this surprising prediction, we analyzed the evolution of the strategy structure. Figure 9.2 shows the effect of promotions on the task strategy structure. The first promotion to this model was the same as in the initial-lockout model (see Figure 8.4): a prepare promotion was applied to the ocular command `o1` resulting in the addition of a new motor command node, `po1`. In this model, however, only *one* event promotion was performed: the movement of `m1` from `pe3` to `pe2`. Since no ocular chain was formed as before, chain promotions applied only to the manual motor chain.

This observation evoked a series of three questions, where each was the logical consequent of the former. The first of these is the obvious question: "Why was `o2` not event-promoted?" The answer is based on several factors. First, as described in Chapter 7, when a command is generated, a promotion suggestion rule will fire for that command (assuming the promotion guidelines are

satisfied), in which case, it will be promoted. The inverse of this is also true and answers this question: `o2` was not promoted because a promotion suggestion was not made, because the `o2` command was never generated.

The next logical question then is "Why was `o2` never generated?" Again, in Chapter 7, the events in the strategy structure were described as preconditions to the motor command; a motor command could not be generated until the event actually occurred. What was not stated there is that the motor command rule has *additional* preconditions that may or may not be satisfied at the time that the event occurs. Recall that `o2` saccades the eye back to the tracking task's stimulus, the cursor. The main precondition of this rule tests whether the eye is already looking at the cursor. If the eye is already looking at the cursor, then the rule does *not* fire.

Since we knew that the `responded` event occurred, we could then deduce that `o2` was never generated because the motor rule itself never matched because the eye was already looking at the cursor. This deduction is confirmed by inspection of the novice trace in Figure 9.3. At cycle 258, the eye was moved to the choice-stimulus. Then at cycle 264, the tracking-task generated a command to move the eye to the cursor. This was done by the `watch-cursor` rule, one of the tracking task's two actions (Figure 6.1). Because the eye was already at or near the cursor at the time the `responded` event occurred, the `o2` rule could not match, therefore it couldn't fire, therefore it could not be promoted.

The final question is: "Why then does this omission cause the poor expert tracking performance observed in Figure 9.1?" The answer lies in the preconditions of the `track-target` rule, the tracking task's rule for moving the joystick to direct the cursor toward the target circle. There are several preconditions but the one of importance here is that the ocular motor processor must not be executing an eye movement command. Referring to the trace, we see that the stimulus appears at cycle 781 and a `FIXATE` command is immediately generated in response. At cycle 785, we see that the `track-target` rule is able to generate its command (in part) because the execution of the fixate has been completed. In other words, the joystick, and hence the cursor is being moved even though the model is looking at the choice stimulus and *not* the cursor.

As a point of comparison, recall the expert performance for the initial-lockout model seen in Figure 8.5. There we see that after the `FIXATE` in cycle 173 is sent, the rule for the pipelined `o2` command immediately fires, moving the eye back to the cursor. Therefore, the behavior pointed out Figure 9.3 is precluded because the ocular motor processor remains busy until the eye has returned to the cursor.

One would expect this behavior to produce poor tracking performance since the model is "not looking at what it is doing". Yet the model predicts very good performance (though it does not qualitatively match the observed data) for the first two-thirds of the conditions. This prediction is due to the fidelity of the EPIC visual sensory processor. Like the human retina, EPIC's simulated retina consists of concentric retinal zones: bouquet, fovea, parafovea, and periphery, in order of increasing eccentricity and decreasing object information quality.

Taking this into consideration, we can deduce that even though the cursor may not be looked at when the `track-target` rule is fired, good tracking performance can still attained because the cursor location information is of sufficient quality in the non-bouquet retinal regions. From Figure 9.1, we see that this appears to be true from zero degrees (superimposed) up to a vertical separation of 22.4 degrees. After this point, there is a pronounced condition effect. We take this to be the point where the cursor location information is no longer of sufficient quality, resulting in errorful tracking behavior.

Thus we have been able to confirm the model's behavior is correct and the promotion learning procedure worked as it was designed to. However, we are left with an unintuitive prediction that a) post-training performance is dependent on the dual-task strategy used during training; and b) an initial-lockout strategy leads to better performance than an initial-interleaving strategy.

NOVICE BEHAVIOR

```
     selected operator: tracking-task & choice-task & executive
210  command: PERFORM PLY generated-by executive
218  command: PERFORM PLY generated-by tracking-task
222  command: PERFORM PLY generated-by tracking-task
226  command: PERFORM PLY generated-by tracking-task
230  command: PERFORM PLY generated-by tracking-task
235  command: PERFORM PLY generated-by tracking-task
239  command: PERFORM PLY generated-by tracking-task
243  command: PERFORM PLY generated-by tracking-task
247  command: PERFORM PLY generated-by tracking-task
251  command: PERFORM PLY generated-by tracking-task
255  command: PERFORM PLY generated-by tracking-task
258  add event: choice-stimulus-onset
258  command: FIXATE PSYCHOBJ382 generated-by executive
259  command: PERFORM PLY generated-by tracking-task
263  command: PERFORM PLY generated-by tracking-task
264  command: MOVE PSYCHOBJ378 generated-by tracking-task
268  recognized choice-arrow
269  verified left-choice-arrow
269  command: PERFORM PLY generated-by tracking-task
269  add event: choice-stimulus-features
270  command: PERFORM PUNCH generated-by choice-task
275  add event: responded-to-choice-stimulus
     finished trial # 4/100
```

EXPERT BEHAVIOR

```
     selected operator: tracking-task & choice-task & executive
750  command: PREPARE PSYCHOBJ440 generated-by executive
756  command: PERFORM PLY generated-by tracking-task
760  command: PERFORM PLY generated-by tracking-task
761  command: MOVE PSYCHOBJ441 generated-by tracking-task
764  command: PERFORM PLY generated-by tracking-task
765  command: PREPARE PSYCHOBJ440 generated-by executive
768  command: PERFORM PLY generated-by tracking-task
772  command: PERFORM PLY generated-by tracking-task
776  command: PERFORM PLY generated-by tracking-task
780  command: PERFORM PLY generated-by tracking-task
781  add event: choice-stimulus-onset
781  command: FIXATE PSYCHOBJ454 generated-by executive
785  command: PERFORM PLY generated-by tracking-task
786  command: MOVE PSYCHOBJ441 generated-by tracking-task
790  recognized choice-arrow
791  verified left-choice-arrow
791  command: PERFORM PLY generated-by tracking-task
791  add event: choice-stimulus-features
792  command: PERFORM PUNCH generated-by choice-task
796  command: PERFORM PLY generated-by executive
797  add event: responded-to-choice-stimulus
     finished trial # 13/100
```

Figure 9.3 Novice and expert traces for the initially-interleaved model

To confirm this prediction, we recalled a study reported by Gopher (1993) that examined the effect of varied task emphasis in dual-task learning. This study consisted of three groups of subjects: in the VP or *varied-priority* group, subjects were required to vary the priorities (as indicated through a visual feedback mechanism) given to the tasks; in the EP or *equal-priority* group, subject were required to give equal priority (as indicated through a visual feedback mechanism) to each task; in the NP or *no-priority* group, subjects were required to give equal priority to each task, but were not given any feedback. It was found that post-training performance was superior for the varied priority (VP) subjects when compared to subjects who were either in the equal-priority (EP) or no-priority (NP) groups.

There appears to be at least a *superficial* connection between these findings and the predictions of our model. First, one can view the initial-lockout model as one where at first, the choice task is given highest priority since it pre-empts the tracking task from the time the stimulus occurred until a response was made. When the model is later switched to use an interleaved, it then gives the tasks equal priority to the tasks in the sense that the tracking task is not disabled.[1] The initial-lockout models uses two priority schemes through training, therefore the initial-lockout model is superficially like the VP group since the model initially emphasized the choice-task then later changed to equal emphasis.

In the initially-interleaved model equal priority was given to each of the tasks, in the sense that the tracking task is not disabled when the choice-task stimulus appears. Since there is no alternative dual-task strategy that allows concurrent performance, they stay with this interleaving strategy. Therefore, the initial-interleaving model is superficially like the EP group which used equal emphasis throughout.

9.2 A RETRAINING REGIMEN CAN IMPROVE PERFORMANCE

Gopher (1993) did not report or speculate on how the relatively poor performance of the equal-priority and no-priority groups could be elevated to reach the superior level of the varied-priority group.

However, our acquisition framework suggests a simple retraining regimen that would enable the initial-interleaving model (which is analogous to the EP group) to reach the performance of the initial-lockout model (which is analogous to the VP group). This could be done by giving the initial-interleaving model (or EP group) some practice trials in the lockout conditions (VP condition). The rationale is that by loading the initial-interleaving post-promotion model, then switching to the lockout strategy and running the model for a few training trials, the promotion learning procedure would perform the necessary promotions on `o2`. Recall that the non-promotion of `o2` was the cause for the difference in performance. After all promotions have ceased, the strategy structure and the procedural knowledge of the model should be exactly the same as the initial-lockout model. By then switching to the interleaving strategy, the model should produce the same level of performance as the initial-lockout model.

This procedure was tested and the predicted performance is shown as the trace labeled *Interleaving + Promotions + Retrained (expert)* in Figure 9.4. Both the reaction time and RMS tracking error measures are essentially the same as the final results for the initial-lockout model shown as the trace *Lockout + Promotions + Interleaving (expert)*.

Notes to Chapter 9

[1] However, at the level of resource utilization, the choice-task still does have priority. Consider the case where the generation of the response to the stimulus co-occurs with the generation of a tracking command. This results in motor processor command conflict. The jam-recovery learning pro-

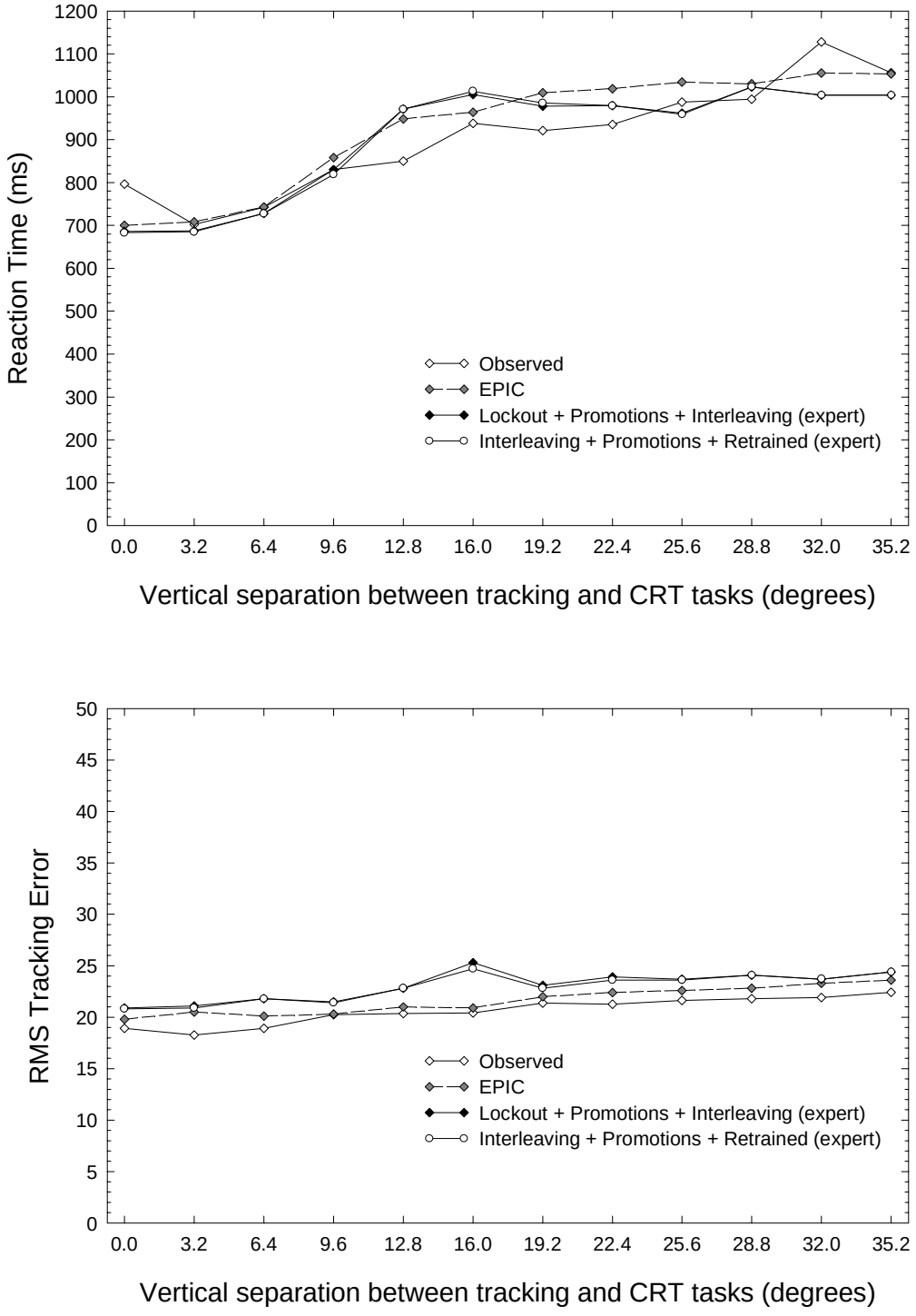


Figure 9.4 Results of retraining the initial-interleaving model

cedure will resolve this conflict by using tasking knowledge which states that choice task commands are preferred over the tracking task commands.

CHAPTER 10

Discussion

10.1 The acquisition framework

The learning procedure performs three styles of promotions: prepare promotions, event promotions, and chain promotions. The three styles were designed to created the three task-independent techniques for improving performance were identified at the beginning of Chapter 7—anticipatory motor programming, movement pre-positioning, and command pipelining, respectively.

The description of the promotion learning procedure in the preceding chapters have no doubt given the impression that promotion styles are applied in a predetermined order: prepare promotions first, followed by event promotions, then chain promotions. This is *not* the case however. We have used an ordered presentation here only to simply the discussion of the effect of each style. The reality is that promotion styles are executed serially in whatever order the promotion suggestions rules fire.

There is a rough dependency between promotion styles however: chain promotions will tend to come after prepare and event promotions. Because prepare and event promotions move commands to earlier events, they can create motor chains by linking the promoted command to the end of a command chain on the previous event, if one existed. A good example of this can be seen in the event-promotion depicted in Figure 10.1 (this is a copy of the event promotion seen in Figure 8.4). There, the ocular and manual commands are both event promoted and eventually are linked to the end of motor chains. (See Appendix C a related discussion of this). When a command is linked in this manner, the link is, by default, a *cautious* link (as defined in Chapter 7 and represented in Figure 7.6). Chain promotions can now be applied to the chain to progressively convert the "cautious" link into an *aggressive* link; a fully pipelined command. Therefore, chain promotions will tend to occur after prepare and event promotions.

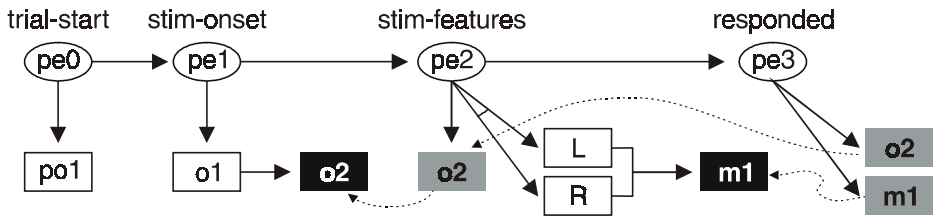


Figure 10.1 Event promotions create motor command chains

The framework has the characteristic that as long as the promotion guidelines of Appendix C are adhered to, the learned models will always produce error-free behavior. (In this context, "error-free behavior" means that the model will always satisfy the task instructions.) This occurs, in part, because the promotion procedure maintains the ordering of commands *within a modality*. Consider Figure 10.1 for example. The `o2` command was promoted two events earlier. When it got to an event where there was already an ocular command, `o2` was appended to the end of that ocular motor chain. This preserves the original ordering between `o1` and `o2`. A similar thing is seen with the `m1` command.

10.2 Switching from lockout to interleaving

With the exception of Chapter 9, this thesis has assumed that novice subjects initially use a lockout dual-task strategy but eventually transition to an interleaved dual-task strategy. Meyer & Kieras (1997a) say that a major contribution of practice on dual-task performance may involve the shift from lockout scheduling to fully interleaved scheduling. Yet, in our dual-task models, the switch from lockout to interleaving required the intervention of the modeler: he had to *manually* add or delete a rule. It appears that the primary reason the switching was performed manually is because it was an unconscious hold-over from the work reported in Chapter 6, where the manual addition of knowledge was the *modus operandi*. A second reason was because manual switching allowed us to easily perform the exploration reported in Chapter 9.

The simple fact is, however, that people transition between the two strategies during practice, and hence, the model should also demonstrate this automatic switch from one strategy to the other. However, we do not yet have a clear understanding of how people make this switch or, more fundamentally, what prompts them to switch at all.

There is one obvious approach for automatically switching dual-task strategies that might be incorporated in the framework: simply switch from lockout to interleaving after all promotions have been performed. We will refer to this as the *post-promotion switching* approach. This simple approach, however, is fraught with problems and complexities.

The first problem with this approach is that is very restrictive. For example, our work on determining the effect of the dual-task strategy initially used on final expert performance (Chapter 9) could not have been performed.

Another problem with the post-promotion switching approach is that is not at all responsive to the goals of the subject. For example, it would only model subjects who started with the lockout strategy and transitioned to the interleaving strategy. Yet it is possible to force subjects to perform using an interleaving (or equal-priority) strategy (Gopher, 1993). It is clear that this approach must be made responsive to the subject's goals. By fixing the responsiveness problems, we would be able to solve the restrictiveness problem discussed above.

But yet another problem is that the post-promotion switching approach assumes that strategy switching is monotonic; i.e., subjects switch to the interleaving strategy and never switch back to the lockout strategy. The reality is that subjects vacillate in the strategies used to perform a task (John & Lallement, 1997). What causes them to switch strategies? It is possible that they are using internal and/or external evaluations of their performance to signal a change in strategy. If so, this provides another point of support that the post-promotion switching approach should be responsive to the subject's goals: the evaluation of a strategy's performance can be used to signal a strategy change.

These three inter-related problems illustrate some of the unexplored issues in modeling the switch between lockout and interleaving, specifically, and the research area of strategy generation, evaluation, and selection in general. A suggestion for further work in this area is given in the Chapter 12.

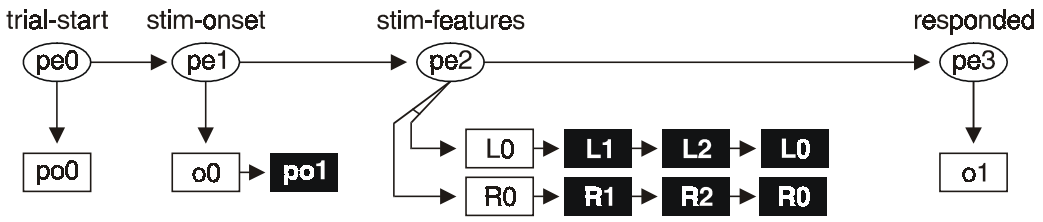


Figure 10.2 The strategy data structure after all *chain promotions*

10.3 Learning rates in Soar

It is worth restating that the focus of this work has only been on producing a transition from novice to expert performance in a manner that is psychologically reasonable. The focus has *not* been on producing predictions of learning rates. The learning rates that have been reported have been on the order of 10 trials, which may be off by as much as two orders of magnitude.

As a result, there is a reflexive tendency by some to question the efficacy of chunking when the issue of learning rates is brought up. Some may think that because this framework does not quantitative predictions of learning rates, chunking must be wrong. We, however, feel that: a) chunking is sufficient for learning, and b) that factors other than chunking have at least as great an influence on learning rates. One such factor is the variability of the situation. Newell (1990) says:

> In general, the rate at which chunking occurs decreases which how much there is to learn. If there's not much to learn, there's not much variability in the situation and the learning curve is fairly steep [the learning rate is high]. . . . If there's lots of detail in the situation, then the learning curve is fairly shallow [the learning rate is low]. At the moment it is unclear where all the variability of detail comes from in the human's learning experience. . . . there are sources of variability in the interior milieu that haven't been accounted for, so it is premature to worry to strongly about the difference in learning rates. But it is an important gap in Soar. . . that needs to be accounted for at the proper time. (pp. 346-48)

During the development of the strategy structure, two design decisions (a euphemism for "shortcuts") was made that, at the time, appeared to be inconsequential. But in light of this discussion, it is clear that the decisions should have been made with greater forethought. These two design decisions are examples of how a disregard for the "variability of the situation" can reduce a framework's predicted learning time. The first decision regards the construction of the strategy structure; the second regards assumptions made in the promotion learning procedure.

In developing the strategy structure, a decision was made to bundle alternative commands together under one node. Revisit the chain promotion procedure as it applied to the example choice-reaction time task—Figure 7.10 is repeated here as Figure 10.2 for convenience. Due to this decision, `L1/R1` were bundled as one node, the same for `L2/R2`, and `L0/R0`. The result of this design decision was that whenever one command was chain-promoted, say `L1`, the effect was the simultaneous promotion of the bundled command, `R1` in this example. The behavioral consequence was the halving of the number of chain-promotions needed to fully pipelined the command chain, which may misrepresents the time to learn the task.

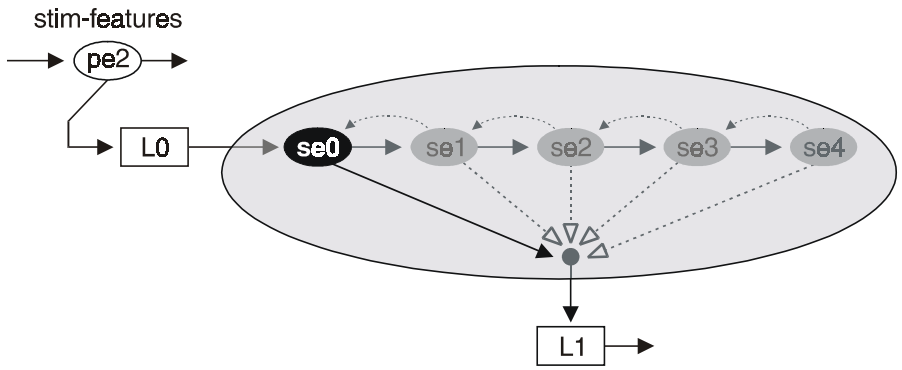


Figure 10.3 Chain promotion of `L1`

In developing the guidelines for the promotion learning procedure, we made the simplifying assumption that learning produced only monotonic changes—there were no demotions—and command were only promoted to the next earliest event—it was not possible to "leapfrog" over the closest event to reach a distant one. Take, for instance, the chain promotions which act on the substructure of motor chains—Figure 7.9 is repeated here as Figure 10.3 for convenience. When a command that is precondition on `se3` is chain-promoted, the new command will always be preconditioned on `se2`; never `se1`; never `se4`.

If the strategy structure were redesigned such that each command node stood alone, the learning time for the example task would easily have been doubled (see Figure 7.11) to roughly 20 trials. If the constraint that promotions must be sequential and monotonic were removed, the framework would exhibit some of the non-monotonic behavior observed in human subject.

Though it is unlikely that either of these changes would suddenly cause the framework to make realistic predictions, this discussion illustrates that models and learning procedures can at times include assumptions and formulations that, in the long run, can significantly misrepresent the learning time predictions.

A few "sources of variability in the interior milieu" of subjects that could account for protracted learning times are: a) motivation—a subject's motivation to perform a task may wax and wain; their interest of lack of interested can affect performance and the rate of learning; b) focus—humans are prone to lapses in concentration while performing a task; c) affect—human emotions can have a positive and/or negative influence on performance; d) cognitive limitations—humans have a limited working memory capacity; humans are also forgetful; e) errors—humans are very prone to errors; f) non-cognitive learning—perceptual and motor learning rates contribute to the overall learning rate.

None of these factors have been taken into consideration in this acquisition framework. We are not aware of any acquisition modeling research has incorporated any of these factors. Finding a principled way of incorporating these varied factor into a modeling environment may be a challenge. Furthermore, many of these factors would have to be architectural additions/modifications; an even greater challenge since careful consideration must be given to how these factor would modulate the activity of the existing architecture.

Another factor that may cause Soar's learning predictions to be too fast is that Soar can "think" too fast. For example, it is possible to build a Soar operator that counts from one to a thousand in a time period representative of 50 ms; one cognitive cycle, or decision cycle, in Soar terminology. It goes without saying that this is unrealistic. If one has a learning procedure that uses large operators

(operators with many sequentially dependent steps), it may be the case that these operators are executing in one decision cycle; 50ms. The learning time might be lengthened if operators were reformulated to take more decision cycles. (See Appendix F for a detailed discussion of this issue and an alternative means of slowing down operator executions.)

The weight of these sources of variability hopefully convinces the reader that chunking, a universal "store" command, is not the sole nor the primary reason that learning rates for this framework are unrealistic.

10.4 Psychological plausibility of the framework

We feel that our framework is psychologically plausible in many way. First, we provide parsimonious initial knowledge to the framework that is plausible in both form and function. The strategy structure is represented declaratively. It is reasonable that subjects have this kind of knowledge after reading task instructions. In addition to the strategy structure, we provide procedural knowledge of representative of the knowledge in the task. This is valid because we assume that subjects are already experts at the individual tasks and they have the necessary task facilitation knowledge to, for example, move eye from one task stimulus to another. Second, our framework learns while performing the task. Finally, we show incremental, multi-trial learning because the learning procedure is constrained by the strategy structure.

The framework is implausible in many ways. The most obvious is the learning rate predictions, as discussed above. Another implausibility is that the learning procedure imposes no cost on task performance, and task performance imposes only a small cost to the learning procedure. This is not psychologically plausible however. Lintern & Wickens (1990) and others who have demonstrated that learning rates can be ill-affected by heavy resource demands. Although we feel that the learning procedures in this work are sound, we believe that further work is needed to explore different forms of interaction between task performance and the learning procedures.

10.5 The contributions of EPIC

In Chapter 3, Newell (1990) was quoted as saying "the theory gives up the constraint on . . . cognition that [perceptual and motor] systems could provide." Having combined EPIC and Soar and done some explorations in learning and performance, what constraints have the perceptual and motor systems (EPIC) imposed on cognition (Soar)?

The most obvious constraint that EPIC has provided are the limitations derived from its perceptual and motor systems. Take, for instance, this simple scenario: an object appear on a screen and cognition "wants to know" what that object is. In EPIC-Soar, cognition will must rely on the visual sensory/perceptual processors and the ocular motor processor. Their impact on the simple task is as follows:

- the visual sensory system models the concentric retinal fields and the increased degradation of information quality with increased eccentricity;
- in order for cognition to clearly "see" an object that is in the periphery, it must send a command to the ocular motor processor to move the eye to the object;
- before the command can be sent however, cognition must ensure that the ocular motor processor is available;
- once the command has been sent, the actual movement of the eye takes time;
- after the eye movement completes, it then takes more time for the information of the object to arrive in cognition;
- additionally, the different features (location, shape, size, color) of the object arrive to cognition at different time.

Without EPIC, it is very unlikely that performance models of perceptual-motor tasks such as this could be successful a performance model could be built in a purely cognitive architecture—such as Soar (Laird, Newel, & Rosenbloom, 1987) or ACT-R (Anderson, 1993). With such architectures, the modeler would have to undertake the non-trivial task of first inventing plausible theories of perception and action, then operationalize those theories. In EPIC, the theories of perception and action are already developed, implemented, tested, verified, and most importantly to the modeler, are architectural. The modeler need only be concerned with the construction of the task model; i.e. the writing of the productions rules for the cognitive processor, or Soar, in the case of EPIC-Soar.

EPIC made other contributions to this work. The acquisition framework is a direct result of the constraints it provides. The idea of motor promotions were in part due to the formulation of the EPIC motor processors as: a) having two semi-independent and sequential phases preparation and execution phases, and b) accepting a PREPARE command.

The chronological task strategy data structure has substructure—the precondition space for chained motor commands (Figure 7.6)—that was derived from both the motor status message and the perceptual (tactile) processor messages. This substructure would not have been apparent without the constraints of EPIC.

At present, EPIC is the only architecture that models both human perception and action with such high fidelity, and hence was the only architecture that could have been used for this work.

10.6 The contributions of Soar

Soar has likewise made many contributions to this work. With its concepts of universal subgoaling and problem spaces, it provided a structured environment for the development of the models and the acquisition framework. Soar's lack of an inherent cognitive bottleneck has been essential to dual-task performance and learning *while* performing, as is done in this framework. Finally, Soar's primary contribution, learning by chunking, has made all the learning in this framework possible. A less tangible contribution is the heritage of successful cognitive modeling work done in Soar (Aasman, 1995; Altmann, 1997; Bauer & John, 1995; Howes & Young, (1996); Lewis, 1993; Lonsdale, 1996, 1997; Miller, 1993; Polk, 1992; Rieman, Young, & Howes, 1996; Wiesmeyer, 1992).

10.7 ACT-R vs. Soar: Is Soar's contribution unique?

There are other architectures that might have been used instead of Soar. Pew & Mavor (1998) compare and contrast eleven architectures for modeling human behavior. Of the eleven, the *only* symbolic learning architecture, other than Soar, is ACT-R (Anderson, 1993).

Could ACT-R have been coupled with EPIC instead of Soar do the work presented in this thesis? To answer this question, we must evaluate ACT-R's ability to: a) support multiple task performance, and b) learn *while* performing multiple-tasks.

First, ACT-R is a multi-match, single-fire production system. As a result, ACT-R manifests an inherent cognitive bottleneck by allowing only one production to be executed per match/fire cycle. In the context of dual-task performance, ACT-R's cognition is a several limited but highly requested resource. It is not clear how ACT-R could concurrently perform `task1`, `task2`, and an executive process. From our experience with both the EPIC and EPIC-Soar models, we have found the unlimited cognitive capacity afforded by a multi-match, multi-fire production system to be necessary in order to perform the tasks. We conjecture that ACT-R could not be generally applied to modeling human performance of complex, high-performance multiple-task combinations.

On whether ACT-R could be used to learn *while* performing, the evaluation must also be in the negative. Our framework learns by creating new procedural knowledge. ACT-R creates new rules through inductive inferences from analogical reasoning from existing procedural knowledge and worked examples. This learning procedure can be thought of as yet another task—a hypothesis of learning as mentioned in Lintern & Wickens (1991)—and thus must competed for access to cognition. Therefore in addition to performing the activities of the constituent tasks and an executive, ACT-R must also perform analogical reasoning. As before, we conjecture that ACT-R could not be used as a general architecture for performance *and* learning since its inherent cognitive bottleneck would prevent it from attending to multiple concurrent activities.

Since ACT-R pursues one activity at a time, it may be possible for a rapid task switching approach to be used, as in Byrne & Anderson (1998). With this commitment, ACT-R would be making the claim that multi-task performance is actually strategically-organized sequential performance. Setting aside the issue of the origin of this strategic organization, a possibly more serious issue is that there phenomena that cannot be easily explained by such an approach. One in particular is *perfect timesharing* (Wickens, 1980; Schumacher, *et al.*, 1997). Perfect timesharing, as defined in Wickens (1991), is a situation in which two tasks, both of a non-trivial difficulty level, are performed concurrently with no decrement, even though each can be shown to interfere with other activities. It is not clear how this phenomena could be addressed when using a rapid task switching approach.

Based on this analysis, the answer to the question, "is Soar's contribution unique", must be that Soar did indeed make a contribution to this work that no other currently available architecture could have.

CHAPTER 11

Contributions

11.1 A learning and performance architecture

Based on the architectural requirements identified in Chapter 3 and illustrated in Figure 3.1, it was found that neither EPIC nor Soar could satisfy all the requirements. In fact, at the time that this work began, there was no learning *and* performance architecture available for the exploration of dual-task acquisition. However, all the requirements were covered by the union of the requirements satisfied by each. The potential for a synergistic merging of EPIC with Soar was obvious and the endeavor was undertaken. The resulting hybrid architecture has been called EPIC-Soar.

This merger was an attempt to get the best of both worlds: the detailed predictions and explanations provided by the perceptual and motor processors of EPIC (an ability Soar does not possess), and the problem solving, planning, and learning capabilities of Soar (an abilities that EPIC does not possess). In Pew & Mavor, (1998), the following has been said about the approach used to create EPIC-Soar:

> This approach of incorporating the mechanism of other architectures and models and "inheriting" their validation against human data promises to result in rapid progress as parallel developments by other architectures emerge (p. 95).

Other than the work presented in this thesis, EPIC-Soar has been used in developing performance models for the Kanfer-Ackerman Air Traffic Controller© task (Lallement & John, 1998a) and for explorations in the role of computational architectures and the modeling idioms used to construct models (Lallement & John, 1998b).

11.2 A taxonomy of executive process knowledge

Recall from Chapter 6 that the executive process knowledge was partitioned into two sets indicating when the knowledge was acquired; either before or after the first dual-task trial (or before/after Stage 2 as depicted in Figure 1.2) was performed. Knowledge that existed before the first trial is called *pre-trial*, while the knowledge that resulted from performance on the task is called *learned*.

Taking into consideration the other knowledge developed in the intervening chapters and our intuitions of the kinds of knowledge subjects may use and/or learn, these two classes can be further elaborated, resulting in an executive process knowledge taxonomy as shown in Figure 11.1. This taxonomy is another contribution of this work. The components of this taxonomy will now be discussed.

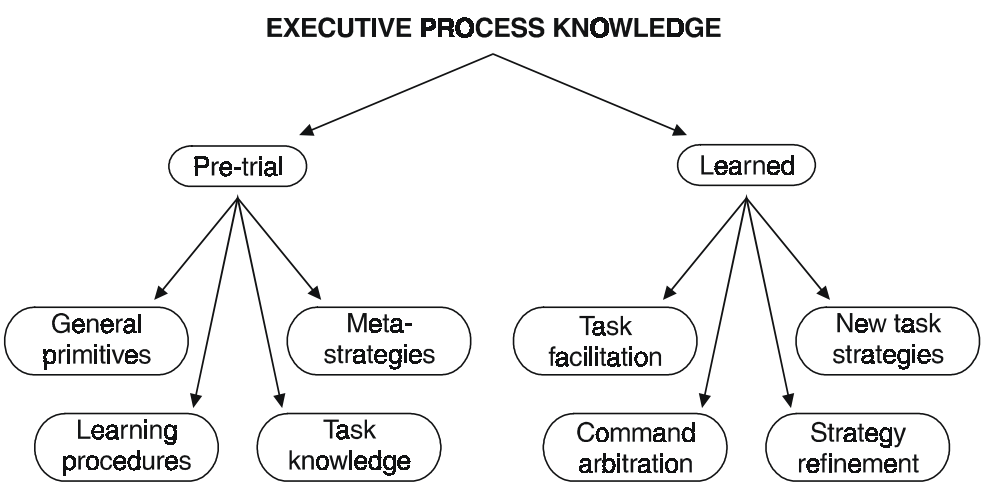


Figure 11.1 A taxonomy of the knowledge learned during dual-task practice

11.2.1 Pre-trial knowledge

The pre-trial class has four subclasses. The *general primitives* class is composed of general primitive actions, such as a `perform-saccade` rule as discussed earlier in Chapter 6, Section 6.5.2. No rules of this class were used in the models developed here, however we expect general primitives could be used with a learning procedure (such as means-ends analysis) to acquire task-specific knowledge.

In the models developed here, the *learning procedures* class consists of both the jam-recovery learning procedure and the promotion learning procedure. Additionally, a means-ends analysis learning procedure might be a member of this class, though it was not used in this work.

The *task knowledge* class consists of task-specific knowledge. Declarative representations of aspects of the task instructions such as task priorities (as used in the jam-recovery learning procedure) and facts are in this class. Declarative representations of the task strategy, such as the chronological task strategy data structure would also be in this class. Any deductions, inferences, or insights about the task that the subject may make based on the instructions would be part of this knowledge class. An example of such cognitions is a pre-performance realization that could result in the creation of the `fixate-on-stim` rule—that when the stimulus appears, the eyes may need to be moved to the stimulus before the choice-task can begin. The *task knowledge* class of course also consists of the procedural knowledge necessary for performing the individual tasks themselves.

At present, only the knowledge that implements the two dual-task strategies—lockout and interleaving—are in the *meta-strategies* class.

11.2.2 Learned knowledge

The 'learned' class also has four subclasses. The *task facilitation* class consists of the knowledge necessary for switching between tasks. The `fixate-on-stim` rule is an example of this kind of rule. Without it, dual-task performance would not be possible. Another example is `EXECUTIVE-put-eye-back-on-cursor-asap` as seen in cycle 55 the trace of the EPIC model (Figure 5.2). This rule returns the eye to the tracking task after it has been moved to the choice stimulus.

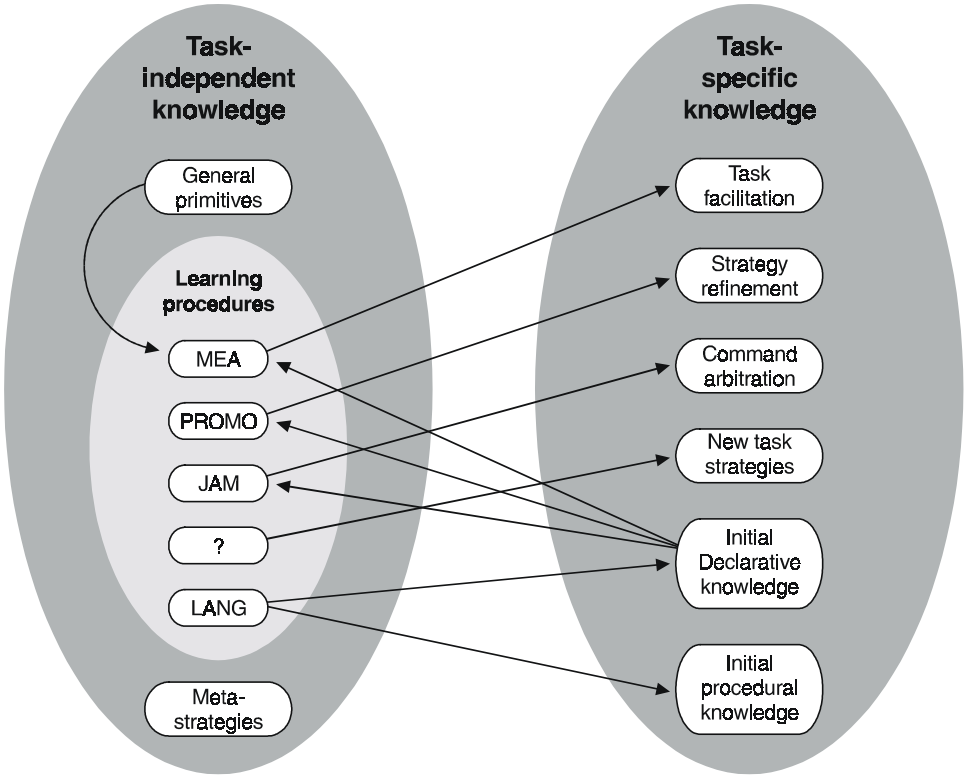


Figure 11.2 The functional relationships of the classes of executive process knowledge

Strategy refinements captures the task-specific knowledge that was created by the promotion learning procedure. This knowledge class contains three subclasses: anticipatory motor programming rules, movement pre-positioning rules, and pipelining rules.

The jam-recovery learning procedure uses task priority knowledge to arbitrate between competing jammed commands. The product of the arbitration is a new rule that in the future will produce the preferred command when the same situation arises where the initial jam occurred. The *command arbitration* class consists exclusively of these rules.

Many researchers have shown that subjects explore task strategies when learning a new task (John & Lallement, 1997; Gordon, et al., 1998; Lee, Anderson, & Matessa, 1995). John & Lallement in particular showed that subjects explored as many as four different strategies for performing their task—the Kanfer-Ackerman Air Traffic Controller© task. It is certainly the case that subjects are creating, using, and evaluating new strategies for the task *while* performing the task. The *new task strategies* class contains these strategies. The as yet unidentified procedures for creating these strategies would be part of the pre-trial *learning procedures* class.

11.2.3 A FUNCTIONAL REPRESENTATION OF EXECUTIVE PROCESS KNOWLEDGE

Figure 11.2 depicts the same knowledge classes as in Figure 11.1, but this time along the dimension of task-specificity. These two classes—task-independent and task-specific—are, with one exception, the same as the pre-trial and learned. The exception is the initial declarative and proce-

dural knowledge, which are part of the pre-trial class in Figure 11.1. This knowledge is task-specific and is hypothesized to be the product of a language comprehension mechanism, possibly similar to the work of Huffman (1994), and is therefore part of the task-specific knowledge class in Figure 11.2.

The figure also depicts the functional relationship between and within the classes. It demonstrates the inputs to certain learning procedures, and shows the products of these learning procedures. The MEA (means-ends analysis) learning procedure uses general primitives to generate task-specific rules such as task-enablement rules. The promotion and jam-recovery learning procedures produce strategy refinement and command arbitration rules, respectively. New task strategies are produced by a yet unidentified procedure. As stated earlier, a language comprehension procedure converts task instructions into declarative and procedural knowledge. The declarative knowledge, deduced or inferred from the task instructions, is used as input to some of the aforementioned learning mechanisms.

Neither of these figures should be taken as complete. Rather they are an attempt to classify the knowledge that was used in the tasks or discussed in this thesis. It is expected that as more dual-task combinations are modeled, other task-independent learning procedure will be found

11.3 An acquisition framework

From the first sentence that introduces Chapter 7, one could say that all this work has been just about learning two rules for the Wickens task: the `prepare-for-stim` and `track-asap` rules. This cannot be denied. However, the work has not been so much about learning those two rules as it has been about *how* to learn those two rules and if the invented framework could be generally applicable to other tasks.

The acquisition framework consists of the following components:

- the jam-recovery learning procedure and the jam-avoidance procedure (Appendix A)
- chronological task strategy data structure (Appendix B)
- the promotion procedure's application guidelines (Appendix C)
- the promotion learning procedure and the promotion suggestion rules (Appendix D)

Admittedly, the promotion learning procedure is a mechanistic, "turn the crank" kind of procedure that blindly implements the promotion guidelines. No "thought", reflection or planning is needed because the promotions are error-proof—the promotions cannot yield behavior that violates the task instructions. In fact, after studying the promotions of the tasks used in the thesis, it quickly becomes obvious how to perform the promotions by hand.

However, while the promotions are now somewhat obvious, they can only be done after the strategy structure has been created. For this reason alone, the strategy structure is possibly the most significant contribution of this work: a proposal for a principled way to represent task strategies for a class of tasks. The promotions are a rather straightforward utilization of the representation to produce learning.

The strategy structure allows a quality of learning that is typically unexpected from all-at-once EBL mechanisms such as Soar's chunking: the procedure produces gradual performance improvement over several trials. This work joins Newell & Rosenbloom (1981) and Miller & Laird (1996) in demonstrating that graded performance is attainable from an all-at-once learning mechanism.

From the first sentence that introduces Chapter 7, one could say that all this work has been just about learning two rules for the Wickens task: the `prepare-for-stim` and `track-asap` rules. This cannot be denied. However, it must be emphasized that these two rules are instances of two task-independent and ubiquitous techniques for improving performance. Additionally, the work has not been so much about learning those two rules as it has been about *how* to learn those

two rules. To address this issue, we have developed a multi-faceted framework that, although designed with multiple-task acquisition in mind, has been demonstrated to be amenable to single-task learning as well, as see in Chapter 7. We are confident that the framework (possibly with a little more elaboration) is general enough to be apply freely to the kinds of tasks that this work addresses—simple perceptual/motor tasks.

11.4 Contributions to other work

When designing a human-machine system, a useful metric for evaluating the proposed design would be a prediction of human performance on the proposed system. Because the system is still in the design phase, empirical evaluation is impossible.

One approach would be to create a performance model in an architecture such as EPIC or Soar. However, in developing such a model, the model builder will make a commitment to one task performance strategy. Given that a wide range of task strategies can be manifest between *and* within subjects, it is clear that a performance model that realizes a single strategy would be of little value in evaluating a human's performance in the proposed system. Building models for each possible performance strategy is certainly not practical.

Kieras & Meyer (1998) have developed a methodology call a *bracketing heuristic* that is used to produce predictive models of human performance that would be useful to a human-machine system designer. The key insight in their methodology is that rather than building models that match a task strategy that may be used by subjects, an easier and more reliable approach is to characterize the *extremes* of the *possible* task strategies.

The methodology consists of three steps. First, define a base strategy for performing the tasks, dictated by the logical requirements of the task. Second, define the *slowest-reasonable* strategy, a version of the base strategy, which consists of nominal adherence to the task requirements. Third, define a *fastest-possible* version of the base strategy, which exploits the capabilities of the architecture to its fullest.

According to the bracketing heuristic, actual human performance on the proposed system should lie somewhere in the envelope defined by the extremes of the slowest and fastest strategies. Thus instead of trying to identify an exact performance strategy, this methodology outlines the boundaries of performance. At present, finding the fastest-possible strategy model involves a potentially lengthy iterative search by the modeler.

The work presented in this thesis resonates with the bracketing methodology in that the slowest-reasonable and fastest-possible strategies can be roughly equated with novice and expert strategies, respectively. The spirit of the bracketing heuristic appears in every performance plot in Chapters 8 and 9. A contribution of this work then is that if given the slowest-reasonable strategy, the acquisition framework might automatically find the fastest-possible strategy.

Take for instance the RMS error graph in Figure 11.3, originally Figure 8.3. The lockout (novice) strategy produced the worst performance and is equivalent to the slowest-reasonable strategy of the bracketing heuristic. After promotions have been applied and the dual-task strategy changed to interleaving, the performance is analogous to the fastest-possible strategy of the bracketing heuristic.

Although Figure 11.3 shows a nicely defined performance envelope bounded by the novice model and the learned expert model, it also shows that the actual performance lies *outside* the envelope. According to the bracketing heuristic, *all* performance should fall within the performance envelope. This brings into question the claim that the framework can be used to find the fastest-possible strategy.

There are two possible explanations however. First, it may be that the difference between the learned model's performance and human performance is not statistically significantly. This could not be evaluated, however, since we did not have variability data for the human data. It may be that

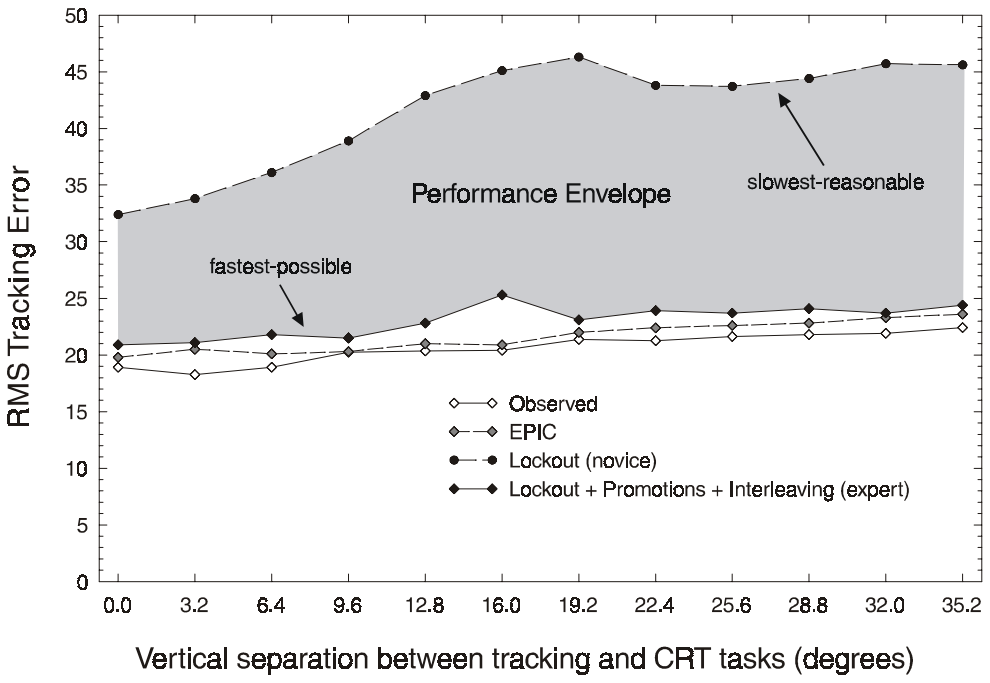


Figure 11.3 The post-promotion model defines the lower bound of the bracketing heuristic

the learned performance and the human data are statistically the same, in which case we could say that the human performance falls within the predicted performance envelope; in fact, humans are performing at the level of the fastest-possible strategy.

If, however, the difference was statistically significant, then two things might be said. First, it may be that our novice Wickens task model or the acquisition framework itself needs revision.

Alternatively, it could also be that the human subjects are simply performing faster than the fastest-possible strategy. Recall that the fastest-possible strategy is defined as: the fastest-possible *version of the base strategy* which exploits the capabilities of the architecture to its fullest. An implicit assumption is that the fastest-possible strategy utilizes the same mental representation of the task as the slowest-possible or base strategies. However, it is possible that a subject may discover a different *mental representation* of the task that could result in a different task strategy. It may then be possible that the new task strategy leads to performance superior to the fastest-possible strategy with the original cognitive representation. In other words, the bracketing heuristic does not take into consideration the metacognitive processes that can impact the actual strategies used, and hence, the final performance.

Therefore, in Figure 11.3, if human performance is indeed significantly better than the learned model's performance, then it may be that subjects were using a mental representation of the task that was not utilized in our task model. While this explanation does not seem likely to apply for our present task—the space of mental representations of this task seem small—we believe it could relevant to more complex tasks.

CHAPTER 12

FUTURE WORK

12.1 FURTHER VALIDATION OF THE TASK MODELS

Further validation of the example CRT task model and Wickens task model can be performed.

In Chapter 7, an example task was modeled. The section that is conspicuous in its absence from that chapter was one that compares both human learning and the final performance results with that predicted by the model and acquisition framework. Before the task model, the framework, and its predictions can be taken seriously, some empirical data for this task must be gathered.

One shortcoming of Wickens task models used in both EPIC and EPIC-Soar is that the tracking task and choice task models were never verified against the *single-task* criterion performance of the subjects. We are relatively confident that the task models we have used are correct. For example, when the choice-task model was run by itself, the average RT was 511ms; a very plausible result. Though the tracking task cannot be similarly evaluated, our intuition is that the two rules—`watch-cursor` and `move-cursor`—define a parsimonious model that is neither too strong nor too weak to produce realistic tracking performance.

Nevertheless, to make the acquisition framework and the post-learning matches and predictions more compelling, we must confirm that there is a pre-learning performance match. In other words, we must confirm that the acquisition framework started with individual task models that are representative of the performance of subjects just before beginning dual-task trials. In dual-task studies, subjects are typically first trained to a criterion level on the individual tasks before beginning dual-task trials.[1]

In addition to verifying the initial models, one should verify the behavior of the expert models with a fine-grained comparison to human behavior. A study could be done to gather eye-tracking data in addition to the normally collected data. The eye tracking data, combined with choice-task response and joystick movements might be used to make this comparison.

12.2 FURTHER VALIDATION OF THE ACQUISITION FRAMEWORK

It was stated in Chapter 1 that this work would only focus on the transition from novice to expert, not on the time to make the transition. As a further validation of the framework, one should compare the learning times (possibly in terms of trials) with an empirical study. It is expected that the framework learns too fast compared to humans.

Related to learning rates, the learning procedure runs concurrently with task performance. There is no cost on task performance due to the learning procedure, and only a small cost to the learning procedure due to task performance. This is not psychologically plausible. Lintern & Wickens (1990) and others who have demonstrated that learning rates can be ill-affected by heavy resource demands. Although we feel that the learning procedures in this work are sound, we

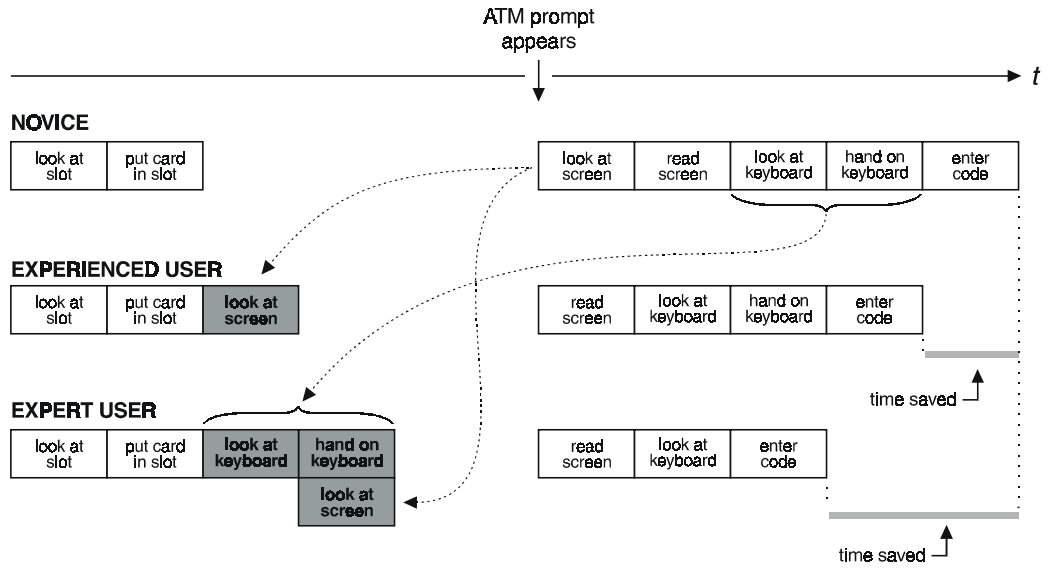


Figure 12.1 An expert strategy that the current framework cannot produce

believe that further work is needed to explore different forms of interaction between task performance and the learning procedures.

In Chapter 10, it was mentioned that the framework assures error-free behavior, in part, because the promotion guidelines maintain the ordering of commands within a modality. However, this order preservation, though ensuring error-free behavior, prevents the framework from finding optimal performance strategies. Recall the ATM task (see Figure 7.3) that was used to illustrate the movement pre-positioning technique. The acquisition framework, if given the novice model depicted there, could easily transition to the experienced strategy shown. However, an expert at the ATM task would produce behavior as shown in Figure 12.1. In addition to pre-positioning the eye, the hand is pre-positioned to the keyboard long before it is necessary. This yields additional reduction in the time to perform the task. The acquisition framework, in its present form, cannot create the expert strategy when given the novice strategy because the promotion guidelines main the ordering of command within a modality. Specifically, the `look-at-keyboard` command could not be promoted before the `look-at-screen` command. To find these globally optimal strategies, the framework will needs further revisions.

12.3 Strategy generation and selection

How do people generate and select strategies? John & Lallement (1997) investigated the evolution of strategy use in the Kanfer-Ackerman Air Traffic Controller © task. They found that subjects explore many different task strategies as they learned to perform the task. Their data also showed that not all subjects converged to the optimal strategy by the end of training and of the few that even found the optimal strategy at some point during training, not all were using it at end of training. These interesting results make one ask how subjects generate strategies, how the generated strategies are selected for use, and how are the selected strategies evaluated. Empirical work on this topic have been performed by Jones & VanLehn (1991, 1994), Reder & Schunn (1996), Schunn & Reder (1996), among others.

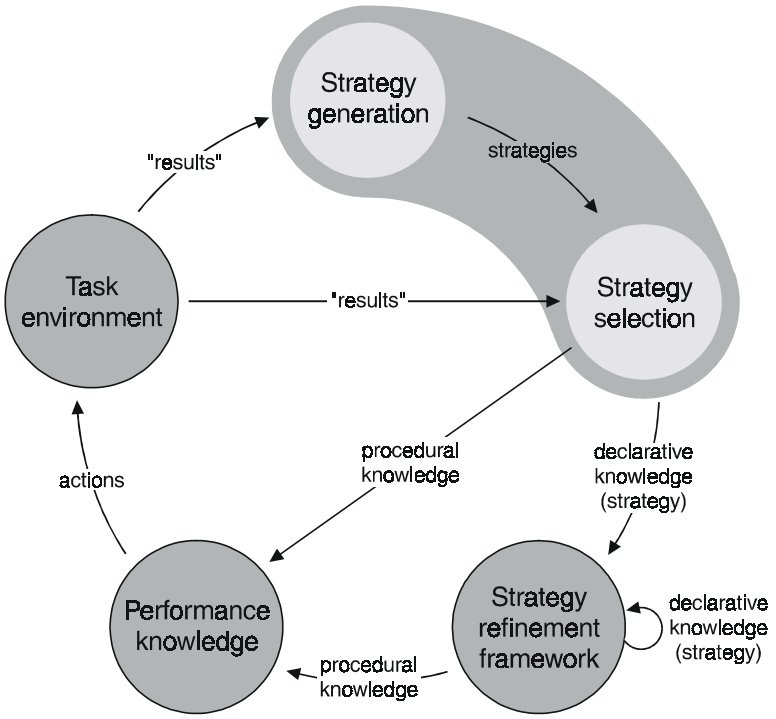


Figure 12.1 A system for realizing gross strategy shifts and fine-grained refinements.

One of the contributions of this work is the proposed structure for representing task strategies. It may be possible that the framework presented here could be built upon to explore the question of how people generate and select task strategies. Figure 12.2 depicts a rough sketch of a system.

The *strategy generator* produces possible strategies for a given task. The generated strategies are passed to a *strategy selection* module which selects a single strategy for exploration. The output of this module is declarative knowledge in the form of the chronological task strategy data structure, and the procedural knowledge to realize the performance implicit in the strategy structure. Though not explicitly shown in the figure, the strategy selection module would also select which meta-strategy—lockout or interleaved—is to be used when performing dual-task combinations.

The *strategy refinement framework* module is the acquisition framework presented in this work. It creates new procedural knowledge (the promoted command rules) and declarative knowledge (changes to the strategy data structure to keep it "in sync" with the procedural knowledge).

The *performance knowledge* is the collection of procedural knowledge for the selected task strategy and any refinements of that strategy. The actions (motor commands) produced by this knowledge are passed to the task environment module. The performance due to the selected strategy is fed back to both the strategy generator and the strategy selector and may result in the generation of new strategies or the selection of a different strategy.

Building such a system would be a major undertaking. Our ideas of how one would proceed are few, though we have an observation: strategies are plans. There is a vast body of work in the area of artificial intelligence that has been done and continues to be done on the topic of plan representation, generation, selection, and evaluation. These algorithms might be used in whole or

part, or they might only provide ideas on which a cognitively-plausible strategy generation and selection system is based.

12.4 Exploring the proceduralization of declarative knowledge

There are two important inputs to the acquisition framework. First, the chronological task strategy data structure, which is declaratively represented, is hand-coded by the modeler and provided to the framework. This input is justified because we believe subjects gain this information primarily from reading task instructions. We also believe this knowledge should declaratively represented.

The second input to the framework is procedural knowledge that performs the task exactly as it is represented in the structure. This input is justified since we assume that we are modeling subjects that are already experts at the individual tasks and have no dual-task experience. We also assume that task facilitation knowledge (the `o1`, `o2`, and `m1` nodes of Figure 8.2) is known by the subjects prior to the first trial of dual-task performance.

An area for future work would be to relax the second set of assumptions by not providing the procedural knowledge. Instead, one could develop a new learning procedure that would learn the procedural knowledge from the declarative strategy structure. There has been much work in this area of the acquisition of skill and knowledge compilation (Anderson, 1983, 1987; Anderson et al., 1981; Neves & Anderson, 1981). There is also prior Soar work by Huffman (1994) that touches on this topic.

If this declarative-to-procedural learning procedure is successful, one could go a step further by not supplying the framework with the chronological task strategy data structure at all. Instead, develop a learning procedure that would accept plain text instructions as input and convert them into a declarative representation; specifically, our startegy data structure. There is previous work in Soar on instructable autonomous agents (Huffman, 1994) and on natural language comprehension (Lewis, 1993). These successful work may be recruited to this effort. There is also non-Soar work in the area of instruction following by Bovair & Kieras (1991) that speaks to this specific issue.

By finding and incorporating such learning procedures, the augmented framework would be able to model a much broader novice-to-expert transition. It would also provide further confirmation that Soar's chunking mechanism is applicable to the wide variety of learning necessary to span this novice-to-expert transition for general task performance

Our intuition is that creating such learning procedures should be quite doable, particularly the procedure for proceduralizing the startegy data structure. We also anticipate that they might be easily accommodated by the framework. However, we have not reviewed and are not familiar with the literature to know what constraints the behavioral phenomena would impose on the construction of such learning procedures.

Notes to Chapter 12

[1] From a personal communication with Christopher Wickens, it appears that subjects were not trained to criterion on the individual tasks. This is understandable since single-task performance levels were not relevant to their study.

APPENDICIES

APPENDIX A

Details of the jam recovery and avoidance learning procedure

In Chapter 6, we mentioned that some of the executive process rules used in the EPIC model of the Wickens task described as defining an explicit, task-dependent strategy for controlling the interleaving of the two tasks. We did not implement this executive process, choosing rather to investigate if perhaps a minimalist control scheme could be used while still producing the same level of expert performance. In this scheme, both tasks are allowed to run concurrently, enabling the steps of each task to apply as they may. The main functional pitfall with this scheme is that there is the risk of two or more motor commands being simultaneously sent to the same modality; a jam condition. Our solution to this problem was to create a task-independent jam-recovery learning procedure. This procedure will be described in this appendix.

A.1 Preference knowledge

The procedure uses task-specific preference knowledge to resolve these conflicts. This knowledge is declaratively and is produced by the firing at system start-up of the production shown in Figure A.1. This rule states that a command generated by the choice-task is preferred over one generated by the tracking-task.

```
sp {task-knowledge*prefer*choice-task-over-tracking-task
  (state <s> ^wickens-state <ws> ^superstate nil)
  (<ws> ^for-subgoal-linking <fsl>)
  -->
  (<fsl> ^task-knowledge <tk> + &)
  (<tk> ^prefer choice-task)
  (<tk> ^over tracking-task)
  (<tk> ^when choice-has-onset)
}
```

Figure A.1 A production for creating the declarative preference knowledge

A.2 The jam-recovery learning procedure

The trace in Figure A.2 starts with the choice stimulus appearing. The expected sequence of actions happen: the eye is moved to the stimulus, the eye is returned to the cursor, a tracking command is sent, the stimulus is recognized and verified. At cycle 364, we see the occurrence of a jam: the choice-task has sent a response at the same time that the tracking task has sent a command to track the target (move the joystick).

All learning (chunking) in Soar arises from activity in a subgoal. Typically, the subgoal will occur naturally but in other cases, the model must be put into a subgoal by artificial means, what one might call "deliberate subgoaling". The jam detection code has been written such that when a jam situation arises (cycle 3725), Soar is deliberately put in a subgoal. At cycle 3727, we see this subgoal.

```
3652:    O: O2450 (choice-task tracking-task executive)
3713: choice-stimulus-onset
3713: command: FIXATE PSYCHOBJ313 generated-by executive
3714:       O: O2487 (wait)
3715:       O: O2487 (wait)
3715: command: PERFORM PLY generated-by tracking-task
3716:       O: O2487 (wait)
3716: command: MOVE PSYCHOBJ252 generated-by executive
3717:       O: O2487 (wait)
3718:       O: O2487 (wait)
3719:       O: O2487 (wait)
3720:       O: O2487 (wait)
3721:       O: O2487 (wait)
3721: command: PERFORM PLY generated-by tracking-task
3722:       O: O2487 (wait)
3723:       O: O2487 (wait)
3723: recognized choice-arrow
3724:       O: O2487 (wait)
3724: verified right-choice-arrow
3725:       O: O2487 (wait)
3725: jammed command: PERFORM PLY generated-by tracking-task
3725: jammed command: PERFORM PUNCH generated-by choice-task
3726:       O: O2487 (wait)
3727:    ==>S: S151 (state no-change)
3728:       O: O2496 (mark-jam)
3729:       O: O2498 (choice-task tracking-task executive)
3729: jammed command: PERFORM PLY generated-by tracking-task
3729: jammed command: PERFORM PUNCH generated-by choice-task
3730:       O: O2498 (choice-task tracking-task executive)
3731:       O: O2499 (identify-jammed-modality)
3732:       O: O2500 (choose-the-preferred-command)
3732: notice: propose*top-level-operator*executive-jam-avoidance
Building chunk-42
3733:    O: O2502 (executive-jam-avoidance)
3733: command: PERFORM PUNCH generated-by choice-task
3734:    O: O2503 (choice-task tracking-task executive)
```

Figure A.2 A trace of the jam-recovery procedure learning a jam-avoidance rule

The action that the model takes when a jam is detected between say `command-A` and `command-B` is as follows: the model stops performing the tasks and instead does some internal reflection/problem-solving to resolve the conflict. When a solution is found, the model produces the solution and resumes performance on the tasks.

During the internal reflection/problem-solving phase, the model deliberately reconstructs the situation that caused the jam, allowing the rules that produced the jam to refire but this time in an internal context. One way to think of this is that when the jam first occurred, the model was behaving reactively (i.e. it was not "paying attention"). By using internal reflection, the model is able to deliberately (i.e. "pay attention") reproduce the jam condition in a manageable context (one where the command is not automatically sent to EPIC but is rather held internally).

After the jam has been recreated, the jam-recovery procedure then applies the preference knowledge described above to decide which action is preferred. For this example, assume that `command-A` is preferred over `command-B`. After that decision has been made, the procedure then created a chunks that says: when the state of the world is such that the rules that produce `command-A` and `command-B` both match, then produce an operator to generate `command-A`.

Returning to the trace, we see that the selected subgoal operator at cycle 3729 is the same as the top-state operator (see cycle 3652). This indicates that the internal state has been reconstructed. At this cycle, we also see that the rules which caused the jam at cycle 3729 have refired. At cycle 3731, a subgoal operator is applied to identify the jammed modality. The operator at cycle 371 does is necessary to ensure that the final learned chunk is correct. At cycle 3732, the preference knowledge is applied to identify which command is preferred.[1] A jam-avoidance chunk is then created. This chunk is an operator proposal and it is immediately applicable as seen at cycle 3733. After it has produced the preferred command, it is terminated and we see the resumption of the dual tasks at cycle 3734.

Figure A.3 shows the resulting jam-avoidance, `chunk-42`. The preconditions of this chunk contain the union of the preconditions of the jamming rules. The action side of the rule is essentially that of the preferred command.

A.3 The application of jam-avoidance rules

In the future, when the situation arises where the same jam would occur, the learned jam-avoidance chunk will fire. This behavior is seen at cycle 57 in Figure A.4. The chunk will propose the `executive-jam-avoidance` operator for the top-state. The existing top-state operator, `(choice-task tracking-task executive)`, is temporarily displaced. In cycle 58, the operator apply and sends the preferred command; the `PUNCH`, thus obviating the impending jam. After the command has been sent, the operator terminates and the previous task operator are reinstated.

A.4 Required inputs to the learning procedure

The only required input is the declarative task preferences rule, as seen in Figure A.1.

Notes to Appendix A

[1] The current jam-recovery procedure only works with two competing commands. This is why choosing the preferred command can be represented by a single operator that takes only one decision (`choose-preferred-command`). To manage jams of more than two commands would require a combinatorial number of comparisons.

JAMMED COMMANDS

```
sp {wickens-task*tracking-task*track-target
   (state <s> ^task-state <ts> ^operator <o>)
   (<o> ^name tracking-task ^io <io>)
   (<ts> ^for-subgoal-copying <fsc>)
   (<fsc> ^task-object <t1>)
   (<t1> ^is-the target)
   (<t1> ^object <obj1>)
   (<obj1> ^physobj <psychobj1>)
   (<fsc> ^task-object <t2>)
   (<t2> ^is-the cursor)
   (<t2> ^object <obj2>)
   (<obj2> ^physobj <psychobj2>)
   (<io> ^input-link <il>)
  -{
    (<il> ^visual.message <vm>)
    (<vm> ^arg1 |GLOBAL-FEATURE|
          ^arg2 |TRACKING-ERROR|
          ^arg3 |SMALL|)
   }
   (<il> ^motor-ocular.message <mom>)
   (<mom> ^arg2 |FREE|)
   (<mom> ^arg1 |EXECUTION|)
   (<il> ^motor-manual.message <mmm>)
   (<mmm> ^arg2 |FREE|)
   (<mmm> ^arg1 |MODALITY|)
   (<io> ^fake-output-link.motor-manual <mm>)
   -->
   (<mm> ^command <c> + &)
   (<c> ^arg1 |PERFORM| +)
   (<c> ^arg2 |PLY| +)
   (<c> ^arg3 |JOYSTICK| +)
   (<c> ^arg4 |RIGHT| +)
   (<c> ^arg5 |ZERO-ORDER-CONTROL| +)
   (<c> ^object1 <psychobj2> +)
   (<c> ^object2 <psychobj1> +)
   (<c> ^generated-for tracking-task +)
   (<c> ^generated-by tracking-task +)
}


sp {wickens-task*choice-task*respond-left-index
   (state <s> ^task-state <ts> ^operator <o>)
   (<o> ^name choice-task ^io <io>)
   (<ts> ^for-subgoal-copying <fsc>)
   (<pe> ^condition right-choice-arrow)
   (<fsc> ^percept-events.percept-event <pe>)
   (<pe> ^name choice-stimulus-features
         ^motor-manual-count 0)
   (<io> ^fake-output-link.motor-manual <omm>)
   (<io> ^input-link.motor-manual <mm>)
   (<mm> ^message <mmm>)
   (<mmm> ^arg1 |PROCESSOR|)
   (<mmm> ^arg2 |FREE|)
   -->
   (<omm> ^command <c> + &)
   (<c> ^update-counter <uc>)
   (<uc> ^motor-manual-count 1)
   (<uc> ^last-motor-manual-count 0)
   (<uc> ^percept-event
          choice-stimulus-features)
   (<c> ^arg1 |PERFORM| +)
   (<c> ^arg2 |PUNCH| +)
   (<c> ^arg3 |LEFT| +)
   (<c> ^arg4 |INDEX| +)
   (<c> ^percept-event-name
         choice-stimulus-features)
   (<c> ^generated-for choice-task +)
   (<c> ^generated-by choice-task +)
}
```

JAM-AVOIDANCE CHUNK

```
sp {chunk-42
    :chunk
    (state <s1> ^superstate nil)
    (<s1> ^executive-state <e1>)
    (<e1> -^terminate-jam-avoidance t)
    (<s1> ^task-state <t1>)
    (<t1> ^for-subgoal-copying <f1>)
   -{ (<f1> -^dual-task-mode t)
      (<f1> -^choice-stimulus-has-onset t)}
   -{ (<f1> -^dual-task-mode t)
      (<f1> ^choice-stimulus-has-onset t)}
    (<f1> ^task-object <t2>)
    (<t2> ^is-the target)
    (<f1> ^task-object <t3>)
    (<t3> ^is-the cursor)
    (<t1> ^for-subgoal-linking <f2>)
    (<f2> ^identified-task-objects t)
    (<f2> ^task-knowledge <t4>)
    (<t4> ^over tracking-task)
    (<t4> ^prefer choice-task)
    (<f1> ^percept-events <p1>)
    (<p1> ^percept-event <p2>)
    (<p2> ^condition right-choice-arrow)
    (<p2> ^motor-manual-count 0)
    (<p2> ^name choice-stimulus-features)
    (<t2> ^object <o1>)
    (<o1> ^physobj <p3>)
    (<t3> ^object <o2>)
    (<o2> ^physobj <p4>)
    (<s1> ^io <i1>)
    (<i1> ^input-link <i2>)
   -{ (<i2> ^visual <v1>)
      (<v1> ^message <m1>)
      (<m1> ^arg3 |SMALL|)
      (<m1> ^arg2 |TRACKING-ERROR|)
      (<m1> ^arg1 |GLOBAL-FEATURE|)}
    (<i1> ^output-link <o3>)
    (<i2> ^motor-manual <m2>)
    (<m2> ^message <m3>)
    (<m3> ^arg2 |FREE|)
    (<m3> ^arg1 |MODALITY|)
    (<m2> ^message <m4>)
    (<m4> ^arg2 |FREE|)
    (<m4> ^arg1 |PROCESSOR|)
    (<i2> ^motor-ocular <m5>)
    (<m5> ^message <m6>)
    (<m6> ^arg2 |FREE|)
    (<m6> ^arg1 |EXECUTION|)
    -->
    (<s1> ^operator <o4> +)
    (<s1> ^operator <o4> =)
    (<o4> ^name executive-jam-avoidance +)
    (<o4> ^type jam-avoidance +)
    (<o4> ^modality-name motor-manual +)
    (<o4> ^preferred-command <p5> +)
    (<p5> ^update-counter <u1> +)
    (<u1> ^motor-manual-count 1 +)
    (<u1> ^last-motor-manual-count 0 +)
    (<u1> ^percept-event
           choice-stimulus-features +)
    (<p5> ^arg1 |PERFORM| +)
    (<p5> ^arg2 |PUNCH| +)
    (<p5> ^arg3 |LEFT| +)
    (<p5> ^arg4 |INDEX| +)
    (<p5> ^percept-event-name
           choice-stimulus-features +)
    (<p5> ^generated-by choice-task +)
    (<p5> ^generated-for choice-task +)
}
```

Figure A.3 The jamming command rules and the resulting jam-avoidance chunk

```
22:    O: O10 (choice-task tracking-task executive)
45: choice-stimulus-onset
45: command: FIXATE PSYCHOBJ346 generated-by executive
46:       O: O18 (wait)
47:       O: O18 (wait)
48:       O: O18 (wait)
48: command: MOVE PSYCHOBJ345 generated-by executive
49:       O: O18 (wait)
50:       O: O18 (wait)
51:       O: O18 (wait)
52:       O: O18 (wait)
53:       O: O18 (wait)
53: command: PERFORM PLY generated-by tracking-task
54:       O: O18 (wait)
55:       O: O18 (wait)
55: recognized choice-arrow
56:       O: O18 (wait)
56: verified right-choice-arrow
57:       O: O18 (wait)
Firing chunk-42
58:    O: O25 (executive-jam-avoidance)
58: command: PERFORM PUNCH generated-by choice-task
59:    O: O10 (choice-task tracking-task executive)
```

Figure A.4 The application of the jam-avoidance chunk

APPENDIX B

The chronological task strategy data structure

This appendix presents the declarative representation of the strategy data structure. The initial and post-promotion structure for the Wickens task will be presented.

B.1 The initial structure

The initial structure is created a production that matches and fires on start-up. This production appears below with descriptions of the important elements.

```
### ------------------------------------------------------------
### chronological task strategy structure for the Wickens task
### ------------------------------------------------------------

sp {elaborate-state*wickens-task-representation
(state <s> ^task-state <ts> ^superstate nil)
(<ts> ^for-subgoal-copying <fsc>)
-->
```

The `^flat-pe-list` structure provides an efficient means to directly access to the perceptual events in the structure.

```
(<fsc> ^instruction <instr>)
(<fsc> ^flat-pe-list <instr>)
(<instr> ^pe-node <penode0> + &)
(<instr> ^pe-node <penode1> + &)
(<instr> ^pe-node <penode2> + &)
(<instr> ^pe-node <penode3> + &)
(<instr> ^pe-node <penode4> + &)
(<instr> ^pe-node <penode5> + &)
```

Likewise, the `^flat-action-list` structure provides an efficient means to directly access to the actions in the structure.

```
(<fsc> ^flat-action-list <actions>)
(<actions> ^action <act10> + &)
(<actions> ^action <act20> + &)
(<actions> ^action <act30> + &)
(<actions> ^action <act31> + &)
(<actions> ^action <act40> + &)
(<actions> ^action <act50> + &)
```

This is the perceptual event node for the `trial-start` event. The perceptual event list is a doubly-linked list, with name pointers to the previous and next events. Since this is the first event, the `^prev-percept-event` name pointer is set to `nil`. The node contains the name of the event, a

link to the actions that are dependent on the event, a count of the number of actions and some miscellaneous counters used the by the promotion learning procedure. At present, there are no actions that are dependent on this node, however, a placeholder `dummy-action` is always present on the event action lists.

```
# event 0
(<penode0> ^percept-event trial-start
           ^num-actions 0
           ^actions <action0>
           ^last-motor-manual-count 0
           ^last-motor-ocular-count 0
           ^prev-percept-event nil
           ^next-percept-event choice-stimulus-onset)
           (<action0> ^action dummy-action + &)
```

For the second event, we see the `^num-actions` is `1`; the action of `fixate-to-stim` rule.

```
# event 1
(<penode1> ^percept-event choice-stimulus-onset
           ^num-actions 1
           ^actions <action1>
           ^last-motor-manual-count 0
           ^last-motor-ocular-count 0
           ^prev-percept-event trial-start
           ^next-percept-event choice-stimulus-features)
           (<action1> ^action <act10> + &)
           (<action1> ^action dummy-action + &)
```

The action structure `<act10>` describes the action. Modality-specific command chains are doubly-linked lists, as seen by the `^modality`, `^prev-action`, and `^next-action` attributes. The first things that is evident about this command is that is an ocular command and it is conditional on whether the features of the stimulus are poor or not—recall that on small vertical separations, it may not be necessary to saccade to the stimulus. If the features are not poor, then no command is sent. If the features are poor however, the a `FIXATE` command is sent. The `^object2 choice-circle` refers to the circle where the choice stimulus will appear. This attribute specifies where the stimulus *will* appear and is used by the prepare promotion style when creating `PREPARE` commands. The `^preconditioned-on` attributes is used exclusively by the chain promotion style. As a command is chain promoted, the `^preconditioned-on` value changes as to chronologically earlier events. The ^version attribute is incremented on various promotions of the command. The value is tested in the motor command rule and it is by this means that old motor command rules are disabled; because they match on an old, expired version number.

```
# event 1: motor 0
(<act10> ^type conditional ^modality motor-ocular ^command <c10> + & ^command <c11> + & ^flag <fn10>
         ^preconditioned-on modality-free ^preparable t ^event-promotable nil ^version 0
         ^object1 onset ^object2 choice-circle ^prev-action nil ^next-action nil)
(<c10> ^condition features-are-poor ^arg1 |FIXATE|)
(<c11> ^condition features-are-not-poor ^arg1 nil)
(<fn10> ^name choice-stimulus-onset ^count 0)
```

The structure of the `choice-stimulus-features` event node is similar to that of the previous, except here, the command is for the motor manual processor. Here, we see the full specification of the motor manual command.

```
# event 2
(<penode2> ^percept-event choice-stimulus-features
           ^num-actions 1
           ^actions <action2>
           ^last-motor-manual-count 0
           ^last-motor-ocular-count 0
           ^prev-percept-event choice-stimulus-onset
           ^next-percept-event responded-to-choice-stimulus)
           (<action2> ^action <act20> + &)
           (<action2> ^action dummy-action + &)

# event 2: motor 0
```

```
(<act20> ^type conditional ^modality motor-manual ^hand |LEFT| ^command <c20> + &
         ^command <c21> + & ^flag <fn20>
         ^preconditioned-on modality-free ^preparable nil ^event-promotable nil ^version 0
         ^object1 nil ^object2 nil ^prev-action nil ^next-action nil)
(<c20> ^condition left-choice-arrow ^arg1 |PERFORM| ^arg2 |PUNCH| ^arg3 |LEFT| ^arg4 |MIDDLE|)
(<c21> ^condition right-choice-arrow ^arg1 |PERFORM| ^arg2 |PUNCH| ^arg3 |LEFT| ^arg4 |INDEX|)
(<fn20> ^name choice-stimulus-features ^count 0)
```

The structure of the `responded-to-choice-stimulus` event node is unique in that it has both a manual and an ocular command chain. The manual command performs a tracking command. Note that it is marked as event-promotable (see Appendix C). This denotes that when this command is generated, an event-promotion suggestion could fire, causing it to be event-promoted. The ocular action structure is similar to that of event 1 in that the command is conditional. This time, it is conditional on the quality of the features of the cursor. Recall that this ocular command returns the eye to the cursor after saccading to the choice stimulus. As before, the vertical separation between the choice stimulus and the cursor may be small enough such that a saccade is not needed, hence this conditionality. This command is also denoted as event-promotable. The `^object1` attribute codes the name of the object that should be looked at, in this case, `cursor`.

```
# event 3
(<penode3> ^percept-event responded-to-choice-stimulus
           ^num-actions 2
           ^actions <action3>
           ^last-motor-manual-count 0
           ^last-motor-ocular-count 0
           ^prev-percept-event choice-stimulus-features
           ^next-percept-event nil)
           (<action3> ^action <act30> + &)
           (<action3> ^action <act31> + &)
           (<action3> ^action dummy-action + &)

# event 3: motor 0
(<act30> ^type unconditional ^modality motor-manual ^hand  |RIGHT| ^command <c30> ^flag <fn30>
         ^preconditioned-on modality-free ^preparable nil ^event-promotable t ^version 0
         ^object1 cursor ^object2 target ^prev-action nil ^next-action nil)
(<c30> ^arg1 |PERFORM| ^arg2 |PLY| ^arg3 |JOYSTICK| ^arg4 |RIGHT| ^arg5 |ZERO-ORDER-CONTROL|)
(<fn30> ^name responded-to-choice-stimulus ^count 0)

# event 3: motor 1
(<act31> ^type conditional ^modality motor-ocular ^command <c31> + & ^command <c32> + & ^flag <fn31>
         ^preconditioned-on modality-free ^preparable nil ^event-promotable t ^version 0
         ^object1 cursor ^object2 nil ^prev-action nil ^next-action nil)
(<c31> ^condition features-are-poor ^arg1 |MOVE|)
(<c32> ^condition features-are-not-poor ^arg1 nil)
(<fn31> ^name responded-to-choice-stimulus ^count 0)
```

The following two events are for the tracking task:

```
# event 4
(<penode4> ^percept-event tracking-error-not-small
           ^num-actions 1
           ^actions <action4>
           ^last-motor-manual-count 0
           ^last-motor-ocular-count 0
           ^prev-percept-event trial-start
           ^next-percept-event nil)
           (<action4> ^action <act40> + &)
           (<action4> ^action dummy-action + &)

# event 4: motor 0
(<act40> ^type unconditional ^modality motor-manual ^hand  |RIGHT| ^command <c40> ^flag <fn40>
         ^preconditioned-on modality-free ^preparable nil ^event-promotable nil ^version 0
         ^object1 cursor ^object2 target ^prev-action nil ^next-action nil)
(<c40> ^arg1 |PERFORM| ^arg2 |PLY| ^arg3 |JOYSTICK| ^arg4 |RIGHT| ^arg5 |ZERO-ORDER-CONTROL|)
(<fn40> ^name tracking-error-not-small ^count 0)

# event 5
(<penode5> ^percept-event cursor-features-not-excellent
           ^num-actions 1
           ^actions <action5>
           ^last-motor-manual-count 0
           ^last-motor-ocular-count 0
```

```
           ^prev-percept-event trial-start
           ^next-percept-event nil)
           (<action5> ^action <act50> + &)
           (<action5> ^action dummy-action + &)

# event 5: motor 0
(<act50> ^type conditional ^modality motor-ocular ^command <c50> + & ^command <c51> + & ^flag <fn50>
         ^preconditioned-on modality-free ^preparable nil ^event-promotable nil ^version 0
         ^object1 cursor ^object2 nil ^prev-action nil ^next-action nil)
(<c50> ^condition features-are-excellent ^arg1 nil)
(<c51> ^condition features-are-not-excellent ^arg1 |MOVE|)
(<fn50> ^name cursor-features-not-excellent ^count 0)

}
```

B.2 The post-promotion structure

After all promotions have been applied, the first event in the structure now has a PREPARE command. The `^object2` attribute codes the name of the task object to use as the argument to the prepare command; `(PREPARE choice-circle)`.

```
# event 0
(P3 ^percept-event trial-start
    ^num-actions 1
    ^actions A7
    ^last-motor-manual-count 0
    ^last-motor-ocular-count 0
    ^prev-percept-event nil
    ^next-percept-event choice-stimulus-onset)
    (A7 ^action A11 ^action dummy-action)

# event 0: motor 0
(A11 ^type unconditional ^modality motor-ocular ^command C15 ^flag F8
     ^preconditioned-on processor-free ^preparable nil ^event-promotable t
     ^object1 onset ^object2 choice-circle ^prev-action nil ^next-action nil)
     (C15 ^arg1 |PREPARE|)
     (F8 ^name trial-start ^count 0)
```

The second event has a two-element ocular command chain. The second command (A6), which returns the eye back to the cursor, has been pipelined unto the first command in the chain. This is evident from the `^preconditioned-on` field begin set to `processor-free`. The value of the `^version` attribute shows that the command has been promoted three times—two event-promotions to move o2 from pe3 to pe2 and then to pe1 followed by a single chain-promotion (since it is an ocular command), as seen in Figure 8.4.

```
# event 1
(P4 ^percept-event choice-stimulus-onset
    ^num-actions 2
    ^actions A8
    ^last-motor-manual-count 0
    ^last-motor-ocular-count 0
    ^prev-percept-event trial-start
    ^next-percept-event choice-stimulus-features)
    (A8 ^action A3 ^action A6 ^action dummy-action)

# event 1: motor 0
(A3 ^type conditional ^modality motor-ocular ^command C2 ^command C3 ^flag F4
    ^preconditioned-on modality-free ^preparable nil ^event-promotable nil ^version 0
    ^object1 onset ^object2 choice-circle ^prev-action nil ^next-action 1)
    (C2 ^arg1 |FIXATE| ^condition features-are-poor)
    (C3 ^arg1 nil ^condition features-are-not-poor)
    (F4 ^name choice-stimulus-onset ^count 0)

# event 1: motor 1
(A6 ^type unconditional ^modality motor-ocular ^command C7 ^flag F7
    ^preconditioned-on processor-free ^preparable nil ^event-promotable t ^version 3
    ^object1 cursor ^object2 nil ^prev-action 0 ^next-action nil)
    (C7 ^arg1 |MOVE|)
    (F7 ^name choice-stimulus-onset ^count 1)
```

The second command on this event, which implements the `track-express` rule, has been pipelined up to first. The value of the `^version` attribute is five—one event promotion to move `m1` from `pe3` to `pe2`, followed by four chain promotions (because it is a manual command), as seen in Figure 8.4.

```
# event 2
(P5 ^percept-event choice-stimulus-features
    ^num-actions 2
    ^actions A9
    ^last-motor-manual-count 0
    ^last-motor-ocular-count 1
    ^prev-percept-event choice-stimulus-onset
    ^next-percept-event responded-to-choice-stimulus)
    (A9 ^action A4 ^action A5 ^action dummy-action)

# event 2: motor 0
(A4 ^type conditional ^modality motor-manual ^hand |LEFT| ^command C4 ^command C5 ^flag F5
    ^preconditioned-on modality-free ^preparable nil ^event-promotable nil ^version 0
    ^object1 nil ^object2 nil ^prev-action nil ^next-action 1
    (C4 ^arg1 |PERFORM| ^arg2 |PUNCH| ^arg3 |LEFT| ^arg4 |MIDDLE| ^condition left-choice-arrow)
    (C5 ^arg1 |PERFORM| ^arg2 |PUNCH| ^arg3 |LEFT| ^arg4 |INDEX| ^condition right-choice-arrow)
    (F5 ^name choice-stimulus-features ^count 0)

# event 2: motor 1
(A5 ^type unconditional ^modality motor-manual ^hand |RIGHT| ^command C6 ^flag F6
    ^preconditioned-on processor-free ^preparable nil ^event-promotable t ^version 5
    ^object1 cursor ^object2 target ^prev-action 0 ^next-action nil)
    (C6 ^arg1 |PERFORM| ^arg2 |PLY| ^arg3 |JOYSTICK| ^arg4 |RIGHT| ^arg5 |ZERO-ORDER-CONTROL|)
    (F6 ^name choice-stimulus-features ^count 1)
```

All the motor commands that were on this event have been event-promoted.

```
# event 3
(P6 ^percept-event responded-to-choice-stimulus
    ^num-actions 0
    ^actions A10
    ^last-motor-manual-count 1
    ^last-motor-ocular-count 1
    ^prev-percept-event choice-stimulus-features
    ^next-percept-event nil)
    (A10 ^action dummy-action)
```

The tracking nodes are the same as before since no promotions applied to those command nodes.

APPENDIX C

Details on the suggestion and application of promotions

This appendix presents the conditions under which promotions are suggestion and once suggested, what action is done to the chronological task strategy data structure.

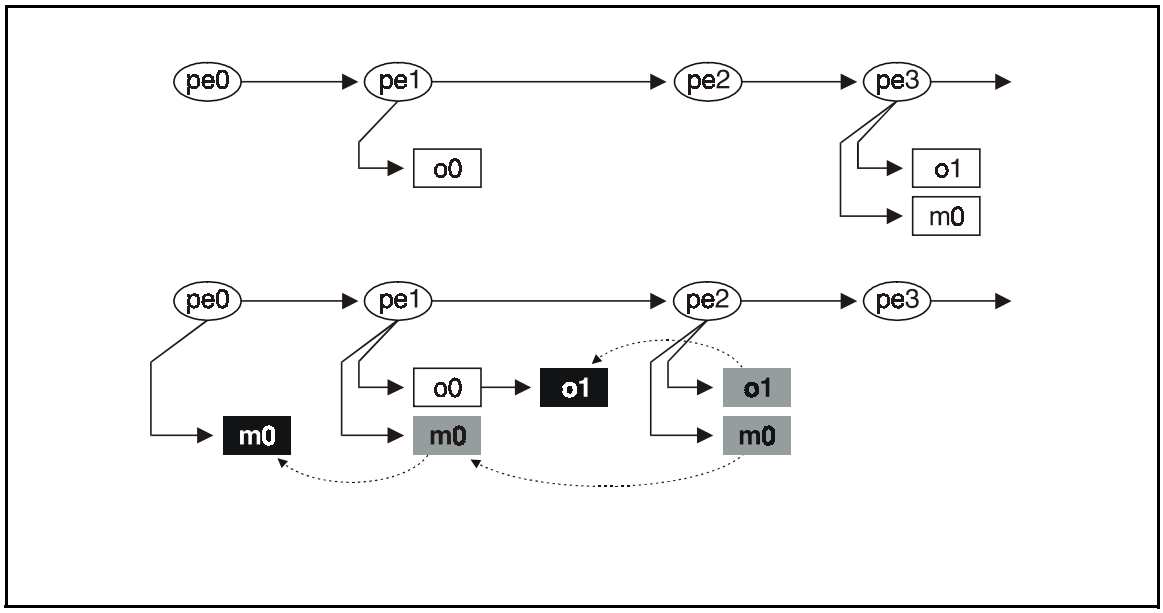
C.1 Event promotion

Event promotions are used to promote motor commands to chronologically earlier perceptual events. It is defined as follows: When an *event-promotable* motor command that is preconditioned on pe-node `pe(t)` is executed, an event-promotion suggestion is generated. A command is *event-promotable* when all of the following conditions are satisfied.

- The command under consideration must be the first command in a motor chain.
- The command is not inextricably bound to the perceptual event to which it was initially dependent. A command is inextricably bound if a promotion of the command would produce performance that deviates from the task instructions. For example, none of the command nodes in Figure 7.5 are event-promotable. Nodes `o0` and `o1` are not event-promotable because the task instructions specifically state or imply that a) the subject should look at the fixation point until the stimulus has onset, and b) looking back at the stimulus should occur after the response is complete. A command can also be inextricably bound due to logical limitations. For example, nodes `L` and `R` in the figure are not event-promotab e manua re pon e annot be ma e unti the timu u feature have arrive a o ing the timu u to be i entifie he event-promotabi it of a omman i in i ate b a `t nil ^event-promotable`

`o1`

`pe(t-1)`



pe(t-1)

o1

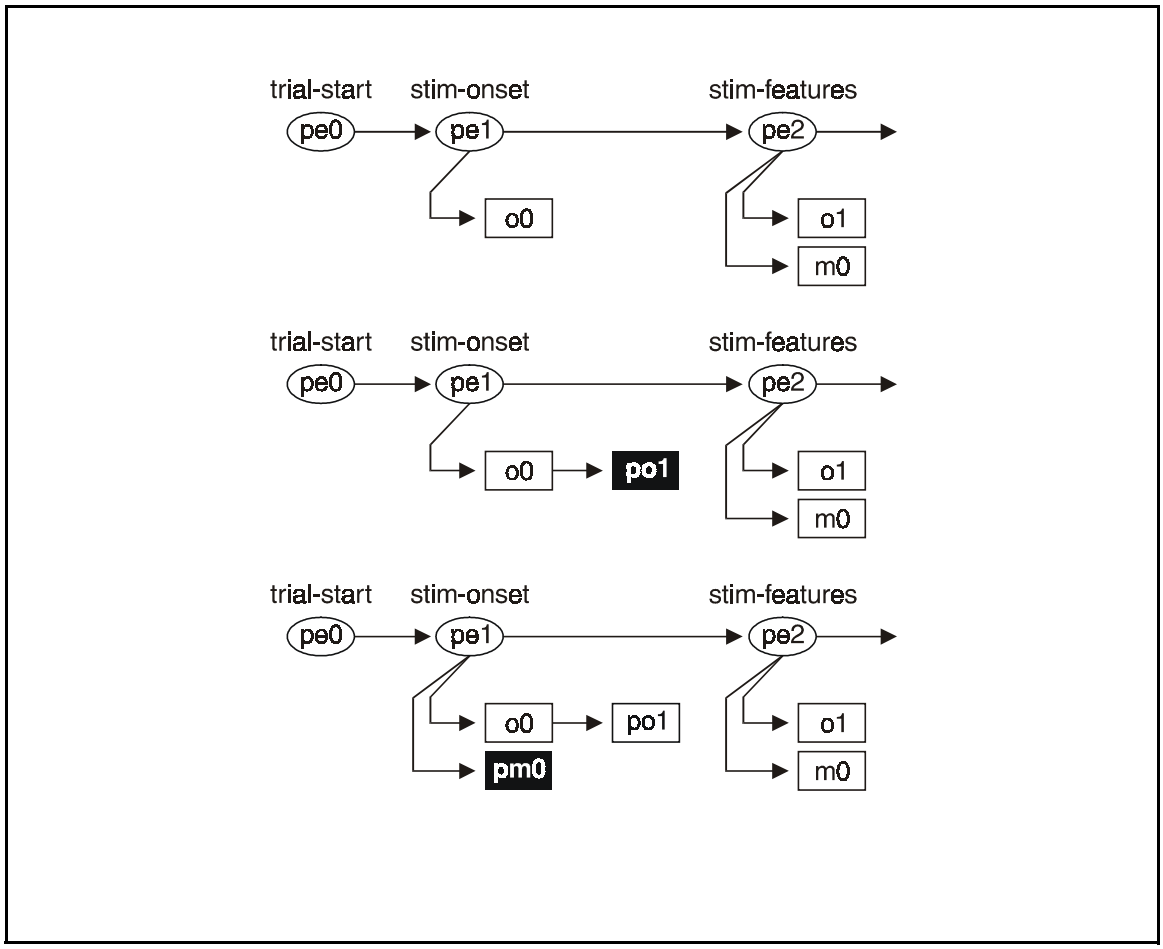
pe2 pe1 o2 pe1
pe(t-1)

o1 pe2 m0

o1 o2
m1

pe(t)

PUNCH



PLY
 ^preparable

trial-start

pe(t-1) (prepare <action>)
<action> pe(t)

pe(t-1)

po1

pe(t-1)

pe(t-1) pm0

se(t)

se(t-1

o0

o0

chunk-1 chunk-3

chunk-2 PRE-

PARE trial-start

executive executive

L1

PREPARE

```
18:    O: O12 (choice-task executive)
71: add event: choice-stimulus-onset
71: command: FIXATE PSYCHOBJ7 generated-by executive
72:       O: O13 (wait)
72: Prepare promotion suggestion: FIXATE choice-circle
73:       O: O40 (copy-suggestion-modality)
74:       O: O57 (copy-suggestion-count)
75:       O: O49 (copy-suggestion-pen)
76:       O: O52 (copy-suggestion-precond)
77:       O: O66 (copy-suggestion-object2)
78:       O: O24 (copy-suggestion-type)
79:       O: O44 (copy-suggestion-cen)
80:       O: O20 (finished-copy)
81:       O: O18 (executive)
82:       O: O18 (executive)
82: recognized choice-arrow
83:       ==>S: S8 (operator no-change)
83: verified left-choice-arrow
83: add event: choice-stimulus-features
84: command: PERFORM PUNCH LEFT INDEX generated-by choice-task
84:          O: O76 (copy-suggestion-type)
85:          O: O118 (copy-suggestion-object2)
86:          O: O96 (copy-suggestion-cen)
87:          O: O92 (copy-suggestion-modality)
88:          O: O109 (copy-suggestion-count)
89:          O: O101 (copy-suggestion-pen)
90:          O: O104 (copy-suggestion-precond)
91:          O: O128 (finished-copy)
92:          O: O127 (prepare-promotion-ocular)
Building chunk-1
Firing chunk-1
Building chunk-2
Firing chunk-2
Building chunk-3
Firing chunk-3
```

L1 R1

po1

^precondition-on

```
94:    O: O12 (choice-task executive)
95:    ==>S: S10 (operator no-change)
95: command: PERFORM PUNCH LEFT MIDDLE generated-by choice-task
96:       O: O129 (wait)
96: Chain promotion suggestion: PUNCH LEFT MIDDLE
97:       O: O187 (copy-suggestion-hand)
98:       O: O189 (copy-suggestion-command-type)
99:       O: O156 (copy-suggestion-arg4)
100:       O: O170 (copy-suggestion-precond)
101:       O: O158 (copy-suggestion-modality)
102:       O: O152 (copy-suggestion-arg3)
103:       O: O140 (copy-suggestion-type)
104:       O: O160 (copy-suggestion-cen)
104: command: PERFORM PUNCH LEFT RING generated-by choice-task
105:       O: O173 (copy-suggestion-count)
106:       O: O136 (finished-copy)
107:       O: O135 (executive)
108:       O: O135 (executive)
109:       ==>S: S12 (operator no-change)
110:          O: O222 (copy-suggestion-precond)
111:          O: O241 (copy-suggestion-command-type)
112:          O: O225 (copy-suggestion-count)
113:          O: O208 (copy-suggestion-arg4)
113: command: PERFORM PUNCH LEFT INDEX generated-by choice-task
114:          O: O210 (copy-suggestion-modality)
114: add event: responded-to-choice-stimulus
115:          O: O192 (copy-suggestion-type)
115: command: FIXATE PSYCHOBJ6 generated-by executive
116:          O: O204 (copy-suggestion-arg3)
117:          O: O239 (copy-suggestion-hand)
118:          O: O327 (copy-suggestion-cen)
119:          O: O359 (finished-copy)
120:          O: O358 (chain-promotion-manual)
Building chunk-4
Firing chunk-4
121:          O: O358 (chain-promotion-manual)
Building chunk-5
Building chunk-6
Building chunk-7
Firing chunk-7
Building chunk-8
Firing chunk-8
```

```
1132: recognized choice-arrow
1133:       O: O2547 (wait)
1133: verified left-choice-arrow
1133: add event: choice-stimulus-features
1134:       O: O2547 (wait)
Firing chunk-65
1134: command: PERFORM PUNCH LEFT INDEX generated-by choice-task
1134: command: PREPARE PSYCHOBJ6 generated-by executive
1135:       O: O2547 (wait)
1135: Event promotion suggestion: PREPARE fixation-point
1136:       O: O2557 (copy-suggestion-type)
Firing chunk-21
1136: command: PERFORM PUNCH LEFT MIDDLE generated-by executive
1137:       O: O2563 (copy-suggestion-arg1)
1138:       O: O2596 (copy-suggestion-object1)
1139:       O: O2583 (copy-suggestion-pen)
1140:       O: O2578 (copy-suggestion-cen)
Firing chunk-41
1140: command: PERFORM PUNCH LEFT RING generated-by executive
1141:       O: O2575 (copy-suggestion-modality)
1142:       O: O2556 (finished-copy)
1143:       O: O2553 (executive)
1144:       O: O2553 (executive)
1145:       ==>S: S96 (operator no-change)
Firing chunk-61
1145: sending command: PERFORM PUNCH LEFT INDEX generated-by executive
1146:          O: O2615 (copy-suggestion-arg1)
1146: add event: responded-to-choice-stimulus
1147:          O: O2627 (copy-suggestion-modality)
1147: command: FIXATE PSYCHOBJ6 generated-by executive
1148:          O: O2609 (copy-suggestion-type)
1149:          O: O2648 (copy-suggestion-object1)
1150:          O: O2630 (copy-suggestion-cen)
1151:          O: O2635 (copy-suggestion-pen)
1152:          O: O2660 (finished-copy)
1153:          O: O2776 (event-promotion-ocular)
Building justification-2
Building justification-3
Building chunk-67
Firing chunk-67
Building chunk-68
Firing chunk-68
Building chunk-69
Firing chunk-69
1154:    O: O12 (choice-task executive)
1155:    ==>S: S102 (operator no-change)
1156:       O: O2780 (copy-suggestion-type)
1157:       O: O2801 (copy-suggestion-cen)
1158:       O: O2806 (copy-suggestion-pen)
1159:       O: O2819 (copy-suggestion-object1)
1160:       O: O2786 (copy-suggestion-arg1)
1161:       O: O2798 (copy-suggestion-modality)
1162:       O: O2833 (finished-copy)
```

```
1163:       O: O2778 (executive)
1164:       O: O2778 (executive)
1165:       ==>S: S104 (operator no-change)
1166:          O: O2839 (copy-suggestion-type)
1167:          O: O2865 (copy-suggestion-pen)
1168:          O: O2845 (copy-suggestion-arg1)
1169:          O: O2878 (copy-suggestion-object1)
1170:          O: O2857 (copy-suggestion-modality)
1171:          O: O2860 (copy-suggestion-cen)
1172:          O: O2892 (finished-copy)
1173:          O: O2891 (event-promotion-ocular)
1174:          O: O2891 (event-promotion-ocular)
Building chunk-70)
```

```
0: ==>S: S1
Created initial strategy structure
1:    O: O2 (ident-objects executive)
Firing chunk-3
Firing chunk-8
Firing chunk-13
Firing chunk-18
Firing chunk-23
Firing chunk-28
Firing chunk-33
Firing chunk-38
Firing chunk-43
Firing chunk-48
Firing chunk-53
Firing chunk-58
Firing chunk-63
Firing chunk-66
Firing chunk-74
Firing chunk-1
Firing chunk-9
Firing chunk-12
Firing chunk-4
Firing chunk-7
Firing chunk-44
Firing chunk-47
Firing chunk-64
2:    O: O2 (ident-objects executive)
Firing chunk-14
Firing chunk-17
Firing chunk-29
Firing chunk-32
Firing chunk-49
Firing chunk-52
Firing chunk-69
Firing chunk-68
Firing chunk-67
3:    O: O2 (ident-objects executive)
3: recognized choice-circle
Firing chunk-19
Firing chunk-22
Firing chunk-34
Firing chunk-37
Firing chunk-54
Firing chunk-57
Firing chunk-71
Firing chunk-73
4:    O: O2 (ident-objects executive)
4: verified choice-circle
Firing chunk-24
Firing chunk-27
Firing chunk-39
Firing chunk-42
Firing chunk-59
Firing chunk-62
```

MOTOR OCULAR PREPARED-FOR FIXATE PSYCHOBJ4

PREPARED-FOR

prepare-for-stim

prepare-for-stim

FIXATE

FIXATE

EXECUTION FREE

EXECUTION BUSY
EXECUTION FREE

EXECUTION FREE

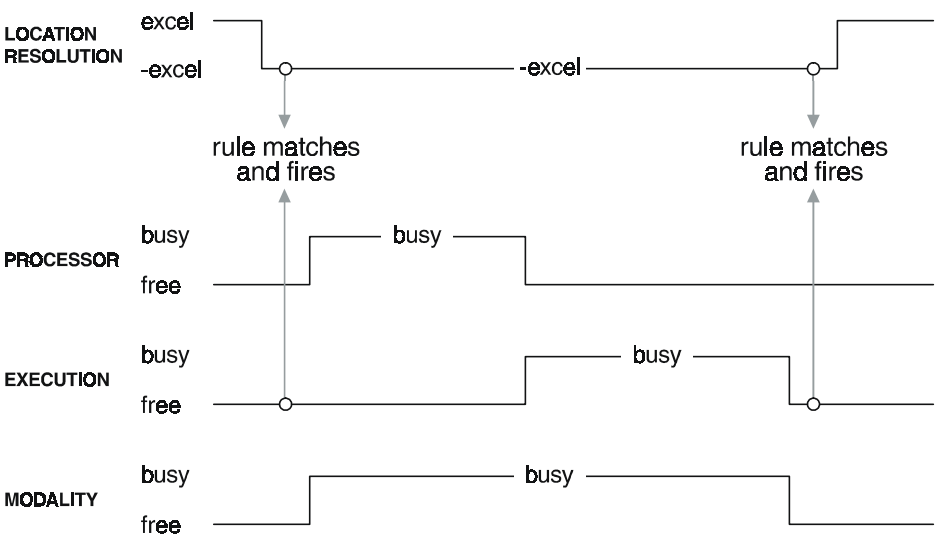
MOTOR OCULAR SPECIAL FINISHED MOVEMENT COMMAND

PROCESSOR EXECUTION

MOTOR OCULAR SPECIAL EYE ALREADY THERE

LOCA-
TION-RESOLUTION EXECUTION
MODALITY BUSY
PROCESSOR BUSY

```
WATCH-OBJECT
   if
      not (visual ?erratic_object location-resolution excellent)
      motor ocular execution free
   then
      send-to-motor ocular move ?erratic_object
```



WATCH-OBJECT

PROCESSOR FREE EXECUTION BUSY
EXECUTION FREE MODALITY FREE
EXECUTION FREE LOCA-
TION-RESOLUTION WATCH-OBJECT

PROCESSOR
EXECUTION FREE LOCATION-RESOLUTION

WATCH-OBJECT
LOCATION-RESOLUTION
LOCATION-RESOLUTION EXCELLENT
EXECUTION FREE

LOCATION-RESOLUTION

EXECUTION LOCATION-RESOLUTION

```
sp {watch-object*apply*move-eye
   (state <s> ^task-state <ts> ^operator.name watch-object ^io <io>)

   . . . . . .

   # are the features not excellent.
  -(<o2> ^location-resolution.value |EXCELLENT|)

   # is the ocular motor processor ready to accept a new command?
   (<il> ^motor-ocular.message <mom>)
   (<mom> ^arg1 |PROCESSOR| ^arg2 |FREE|)

   # this test will delay any refirings until we know what the
   # features have had a chance to be updated in working memory
  -(<il> ^still-waiting-for location-resolution-of-the-object)

   (<io> ^fake-output-link.motor-ocular <mo>)
   -->
   (<mo> ^command <c> + &)
   (<c> ^arg1 |MOVE| +)
   (<c> ^object1 <psychobj> +)
   (<c> ^wait-for-this-feature-to-change |LOCATION-RESOLUTION| +)
   (<c> ^LHS-will-test location-resolution-of-the-object +)
   (<c> ^feature-delay 2 +)
}
```

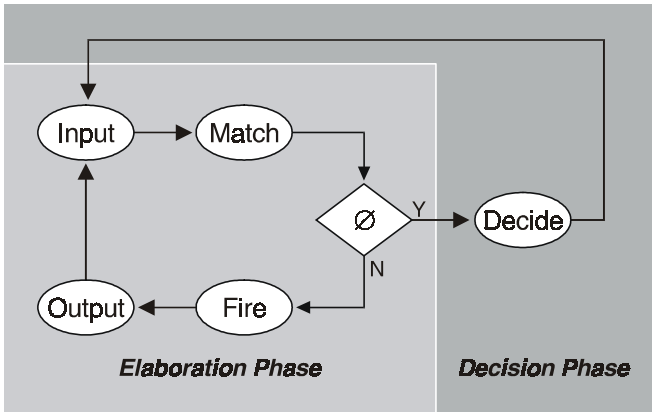
LOCATION-RESOLUTION LOCATION-RESOLUTION

LOCATION-RESOLUTION
LOCATION-RESOLUTION

feature-delay
LOCATION-RESOLUTION
location-resolution-of-the-object
(<il> ^still-waiting-for location-resolution-of-the-object)

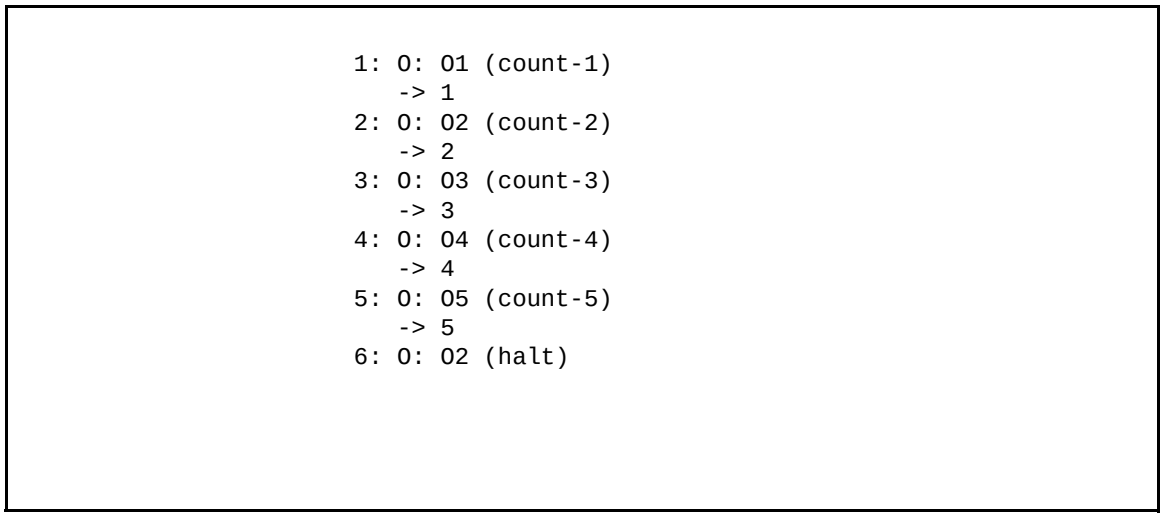
feature-delay

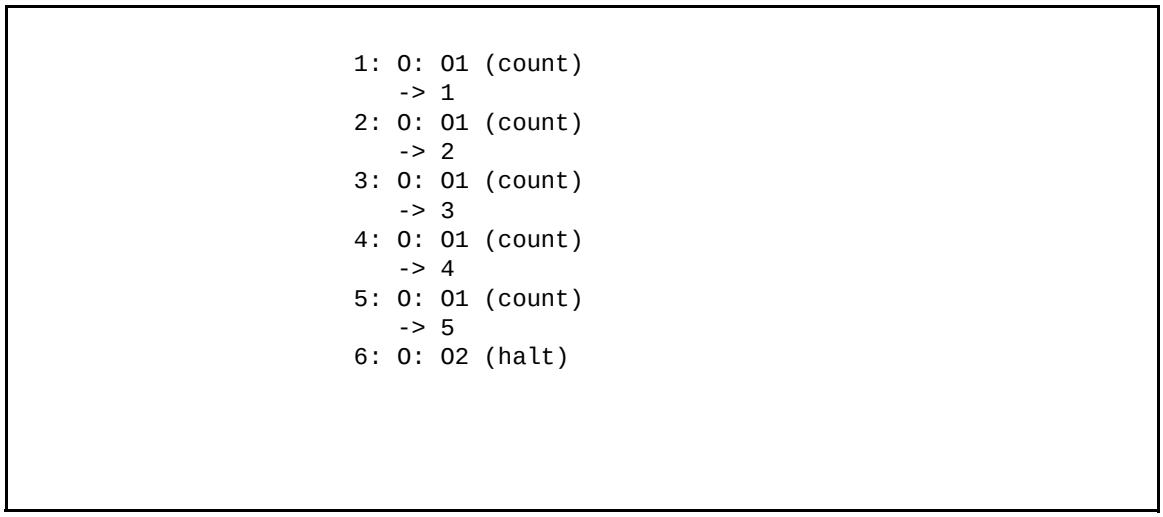
MOTOR OCULAR SPECIAL FINISHED MOVEMENT COMMAND
feature-delay MOTOR OCULAR SPECIAL EYE ALREADY THERE

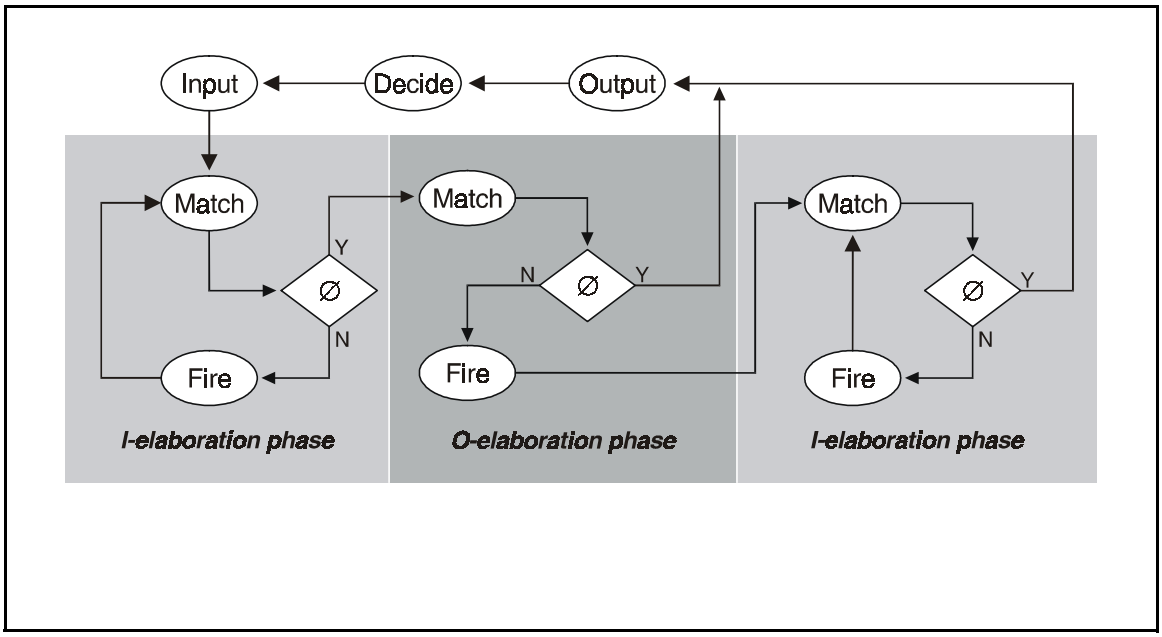

Input
Match
Ø
Y
Decide
N
Output
Fire
Elaboration Phase
Decision Phase

```
sp {counter*apply*1                    1: O: O1 (counter)
   (state <s> ^operator <o>)                 1 -> 2 -> 3 -> 4 -> 5
   (<o> ^name counter)                 2: O: O2 (halt)
  -(<o> ^count 1)
-->
   (<o> ^count 1 +)
   (write |     1|)
}

sp {counter*apply*2
   (state <s> ^operator <o>)
   (<o> ^name counter)
   (<o> ^count 1)
-->
   (<o> ^count 1 -)
   (<o> ^count 2 +)
   (write |  -> 2|)
}
```


1: O: O1 (count-1)
-> 1
2: O: O2 (count-2)
-> 2
3: O: O3 (count-3)
-> 3
4: O: O4 (count-4)
-> 4
5: O: O5 (count-5)
-> 5
6: O: O2 (halt)


1: O: O1 (count)
-> 1
2: O: O1 (count)
-> 2
3: O: O1 (count)
-> 3
4: O: O1 (count)
-> 4
5: O: O1 (count)
-> 5
6: O: O2 (halt)


Input
Decide
Output
Match
Y
∅
N
Fire
I-elaboration phase
Match
N
∅
Y
Fire
O-elaboration phase
Match
∅
Y
N
Fire
I-elaboration phase