Threaded Cognition: An Integrated Theory of Concurrent Multitasking

Dario D. Salvucci
Drexel University

Niels A. Taatgen
Carnegie Mellon University and
University of Groningen

Please address correspondence to:

Prof. Dario Salvucci
Department of Computer Science
Drexel University
3141 Chestnut St.
Philadelphia, PA 19104
Phone: 215-895-2674
Fax: 215-895-0545
Email: salvucci@cs.drexel.edu

Running head:

Threaded Cognition

In Press, ***Psychological Review***

August 6, 2007

Abstract

We propose the idea of "threaded cognition," an integrated theory of concurrent multitasking — that is, performing two or more tasks at once. Threaded cognition posits that streams of thought can be represented as threads of processing coordinated by a serial procedural resource and executed across other available resources (e.g., perceptual and motor resources). The theory specifies a parsimonious mechanism that allows for concurrent execution, resource acquisition, and resolution of resource conflicts, without the need for specialized executive processes. By instantiating this mechanism as a computational model, threaded cognition provides explicit predictions of how multitasking behavior can result in interference, or lack thereof, for a given set of tasks. We illustrate the theory in model simulations of several representative domains ranging from simple laboratory tasks such as dual-choice tasks to complex real-world domains such as driving and driver distraction.

Keywords: Threaded cognition, multitasking, executive control, dual-task performance, perceptual refractory period, driving, cognitive architectures

Introduction

One of the most impressive aspects of the human cognitive system is the ability to manage and execute multiple concurrent tasks. Concurrent multitasking (which we will often refer to as simply "multitasking") is a ubiquitous phenomenon in our daily lives, from work to leisure to everyday routine. In some situations, multitasking can seem nearly effortless (e.g., walking and talking); for other situations, it can seem extremely difficult if not impossible (e.g., reading and listening to two distinct sentences); for still others, multitasking performance may depend heavily on the individual and/or the environment (e.g., singing while playing an instrument, or dialing a phone while driving). This space of possibilities raises an enormous challenge in understanding, on the one hand, the human system's astonishing capacity for multitasking, and on the other hand, the sometimes severe limitations on multitasking performance. In this paper, we propose a new theory called *threaded cognition* that provides a theoretical and computational framework for understanding, modeling, and predicting performance during the concurrent execution of arbitrary tasks.

Meyer and Kieras (1997) provide an in-depth historical background of the study of multitasking performance, dating back several decades (e.g., Telford, 1931; Craik, 1948; Welford, 1952) and covering the vast literature concerning the potential for processing bottlenecks in various stages of perceptual, cognition, and motor processing (e.g., Broadbent, 1958; Keele, 1973; Pashler, 1984). Such studies and theories emphasized the role of various processes as representative of distinct resources in which bottlenecks may or may not appear (depending on particulars of the task domain). Further work codified these efforts into related but distinct theories of processing on multiple resources. Navon and Gopher's (1979) multiple-resource theory specified how tasks using separate resources may proceed simultaneously without interference, but in the presence of resource conflicts, the required resource can allocate part of its processing time to each task. Wickens (2002; see also 1984) posited an alternate theory of multiple resources that accounts for performance using the four resource "dimensions" of processing stages, perceptual modalities, visual channels, and processing codes. Both efforts attempted to unify the many empirical findings related to performance bottlenecks into more comprehensive frameworks with which to reason about and quantify resource needs and potential task interference.

In relating resources and bottlenecks to the larger issue of cognitive control, several theorists have posited the need for executive processes that manage and schedule individual tasks. Norman and Shallice (1986) described a two-component system that performs contention scheduling for basic routine action but requires a supervisory attentional system for higher-level control. Baddeley (1986) defined a "central executive" responsible for management and selection of processing routines in the context of working memory. Such theories have been influential in broadening our understanding of multitasking by considering the processes underlying multitasking behavior and how these processes contend for and utilize available resources. At the same time, there has long been a recognized desire for further specification of the mechanisms that embody these executive processes — in order to, as has been said, banish the "homunculus" of cognitive control (see, e.g., Altmann, 2003; Logan, 2003; Monsell & Driver, 2000).

With this desire in mind, many theorists have recently turned to computational modeling as a methodology for providing rigorous specifications of both executive and task processes. One body of work has examined multitasking performance in smaller-scale laboratory tasks, modeling and accounting for various effects of, for example, stimulus or response modality and task difficulty in dual-choice tasks (e.g., Byrne & Anderson, 2001; Meyer & Kieras, 1997-b; Taatgen, 2005). Another body of work has modeled multitasking behavior as it occurs in particular complex task domains — for example, models of fighter piloting (Jones et al., 1999), driving (Salvucci, 2005, 2006), air-traffic control (Taatgen & Lee, 2003), and similar domains (see Gluck & Pew, 2005, and Gray, 2007, for further examples). Still other work has focused on computational models of executive control in routine activities (Cooper & Shallice, 2000), which could in principle serve as a general multitasking mechanism; however, these models currently provide no theory to guide the specification or learning of higher-level schemas required to resolve resource conflicts, and thus have not been applied to modeling concurrent multitasking. Overall, this range of work has demonstrated the many benefits of a computational modeling approach, including a finer-grained analysis of behavior over time and the mapping of behavior to quantitative measures that can be compared directly to human performance data. However, these efforts have almost exclusively utilized domain-specific mechanisms of multitasking fine-tuned for a single task domain, making it difficult to generalize the models and mechanisms to other (even closely related) task domains.

In summary, more general theoretical treatments such as those of Norman and Shallice (1986) and Baddeley (1986) do not provide sufficient detail for instantiation as computational mechanisms, whereas current computational models do not provide sufficient breadth to generalize across task domains. We wish to achieve the best of both approaches — that is, to develop a general, domain-independent model of multitasking as a fully specified computational mechanism. In one such effort, Kieras, Meyer, Ballas, and Lauber (2000) (see also Kieras, 2007) developed models in their EPIC cognitive architecture (Meyer & Kieras, 1997) to explore the challenges of unifying domain-specific "customized executives" into a single domain-independent "general executive." They developed customized executives for several tasks, and for one task in particular (tracking and choice, detailed later), also developed two versions of a general executive and compared their predictions to those of the customized executive. However, they found that neither version of their general executive could adequately account for human behavior in this task, and resorted to a customized executive to achieve their best account of the human data.

In our view, approaches that require supervisory or executive processes to manage multitasking behavior have a significant limitation in that they do not adequately account for multitasking between two arbitrary tasks. For example, a person may be well practiced in skills such as washing dishes or performing mental arithmetic, but may never have performed both tasks together. Nevertheless, even in the first attempt, a person can perform such tasks concurrently when asked. Any approach that requires executive processes must account for this ability either by defining a general executive or by defining how specialized executives for the two tasks can be learned. We believe that humans have a basic ability to perform multiple concurrent tasks, and that this ability does not require supervisory or executive processes. Instead, this ability is best represented by a general, domain-independent, parsimonious mechanism that allows for concurrent processing and provides basic resource conflict resolution (see Liu, Feyen, & Tsimhoni, 2005). While higher-order planning may still arise in deliberate reasoning about one's own multitasking behavior, it is this basic ability that allows us to perform multiple concurrent tasks as often arises in everyday behavior.

In this paper, we propose a theory of *threaded cognition* that provides both a conceptual theory and an associated computational framework for general domain-independent multitasking. Threaded cognition posits that cognition maintains an active set of task goals that result in *threads* of processing across a set of available resources. Specifically, it posits that threads are coordinated by a serial procedural resource that combines inputs from other resources (e.g., perceptual and motor resources) and initiates new processing on these resources. The theory specifies how threads acquire and release resources and how conflicts with respect to acquisition of the central procedural resource are resolved. As such, the theory claims that concurrent multitasking does not require supervisory executive processes; instead, concurrent multitasking emerges from the interaction of autonomous process threads in conjunction with a straightforward mechanism for resource acquisition and conflict resolution. The computational instantiation of this mechanism, based in the ACT-R cognitive architecture (Anderson et al., 2004), allows multiple models of arbitrary tasks to be executed together, generating immediate predictions of the resulting multitasking behavior. As such, threaded cognition provides a domain-independent theory and framework for understanding, representing, and predicting multitasking performance.

Threaded Cognition

The core idea of threaded cognition is that multitasking behavior can be represented as the execution of multiple task *threads*, coordinated by a serial cognitive processor and distributed across multiple processing resources. In this section we formalize our notion of threaded cognition and threads in two sets of core assumptions. The first set of assumptions lays the theoretical groundwork by presenting an underlying framework for single-task performance. The second set of assumptions extends the single-task assumptions into our broader theory of threaded cognition and multitask performance. In all, threaded cognition represents a consolidated effort to combine our two recent individual approaches to multitasking (Salvucci, 2005; Taatgen, 2005) into a single more cohesive account.

Before delving into the details of threaded cognition, we first wish to develop an intuitive feel for the theory by drawing an analogy — namely, that human cognition, particularly aspects related to multitasking and executive control, behaves much like a cook preparing food in a kitchen. A cook (in any variety, from a restaurant chef to a short-order cook to a parent cooking for a family) manages a number of resources in order to complete a stream of orders. Let us imagine a cook at a cooking station, illustrated in Figure 1(a), with several available resources: an oven for baking, a stove and pot for boiling, and a mixer for mixing. Let us also imagine that orders are received on individual slips of paper that indicate the desired dish. For a given dish, the cook must execute a well-practiced sequence of steps to achieve the desired results. For instance, to prepare fish, pasta, or a cake, a cook might execute the (grossly simplified) steps shown in Figure 1(b). Importantly, each step generally involves the cook initiating some process (e.g., boiling or baking), waiting for the process to complete, and then moving on to the next step. Each step is thus associated with both the desired food order and the current state of the various resources.

<< Insert Figure 1 approximately here >>

Now, imagine that the cook must work to fill multiple orders simultaneously — for example, to prepare all three dishes in the figure at the same time. On the one hand, some processes can clearly execute in parallel; for instance, the fish can bake in the oven while the pasta boils on the stove. On the other hand, several constraints limit the amount of parallelism in filling orders. First, the cook may experience a resource conflict where two or more orders require the same resource, such as the fish and cake orders that require the oven at the same time (let us assume with different temperatures). Second, the cook him/herself is sometimes the bottleneck, specifically when processes for two or more orders require attention from the cook at the same time — for instance, when the oven-preheating and the water-boiling processes end simultaneously. In this case, one order must be delayed while the cook handles the next step for the other order. Thus, the cook can often allow multiple orders to proceed in parallel, but conflicts for either the cook or another resource sometimes reduce parallelism and lead to delays in order processing, shown in gray in the figure. In the example, the fish order experiences no delays; the pasta order experiences slight delays because of contention over the cook; and the cake order experiences large delays because of contention for the oven. We might even imagine that the cook requires a cookbook for an unfamiliar order, in which case frequent accesses to the cookbook would result in further delays.

In threaded cognition, the central procedural resource can be analogized to the cook, and other resources (e.g., perceptual and motor) can be analogized to the various cooking resources available to the cook. The procedural resource, like the cook, collects processing results from resources and redirects this information to make new processing requests to other resources; goals, rather than orders, guide the processing to directed action in service of a desired task. Conflicts can arise either when multiple tasks require the same peripheral resource (e.g., when two tasks both require vision), or when multiple tasks require attention from the central procedural resource (e.g., when auditory and visual tasks finish encoding simultaneously and require the procedural resource to proceed). And just as a cook may require a cookbook in the initial stages of learning, threaded cognition initially relies on memorized instructions, but transforms a skill to a more highly proceduralized process through learning. In essence, the interplay of resource parallelism with potential for resource conflicts gives rise to a rich array of possible multitasking scenarios that demonstrate both the power and limitations of human multitasking. We now define the theory of threaded cognition more formally in the two sets of core assumptions.

Core Assumptions for Single-Task Performance

In developing our theory of multitask performance, we wish to start with a firm theoretical grounding for single-task performance that we can then extend to multitask performance. The theoretical arguments in the following assumptions draw heavily from theories of resource allocation and utilization (e.g., Navon & Gopher, 1979), attention and control (e.g., Norman & Shallice, 1986), production systems (e.g., Newell & Simon, 1972), and computational cognitive architectures (e.g., Meyer & Kieras, 1997). In addition, to make the theory concrete as a computational formalism, we adopt the representations and mechanisms of the ACT-R cognitive architecture (Anderson et al., 2004) along with various computational treatments that have extended the architecture in ways relevant to multitasking (e.g., Byrne et al., 2001; Salvucci, 2001-b; Taatgen & Lee, 2003; Taatgen, van Rijn, & Anderson, 2007).

Processing Resources Assumption*: Human processing resources include cognitive, perceptual, and motor resources.*

Our first core assumption sets the groundwork for the various processing facilities, or "resources" (see, e.g., Navon & Gopher, 1979; Norman & Bobrow, 1975), available to the human system. These resources can be broadly characterized as related to the cognitive, perceptual, and motor systems. The perceptual resources acquire information from the external world, and include a variety of systems that facilitate visual, auditory, tactile, olfactory, and other types of perception. The motor resources enable the body to perform actions in the external world, as a response to stimuli or sometimes to facilitate perception (e.g., eye movements for visual perception or hand movements for tactile perception). The cognitive resources process the information entering the perceptual resources and guide further perception and motor action based on both external situations and internal state. This broad assumption is, in essence, nothing new, but follows a long line of research on processing bottlenecks (e.g., Broadbent, 1958; Pashler, 1994), theories of resource integration and interference (e.g., Wickens, 1984, 2002), and cognitive architectures (Anderson et al., 2004; Meyer & Kieras, 1997).

In our computational modeling of these resources, we will make the assumption that a resource has two components: a *module* that performs the processing related to that resource, and one or more *buffers* that serve as a communication channel between the module and (as we shall soon describe) the procedural resource. For example, a request to the visual resource to encode a visual stimulus would evoke processing in the visual module, and when completed, the result — that is, a representation of the encoded stimulus — would be placed in the visual buffer and thus be available for subsequent processing. We shall further define the individual resources in the assumptions that follow.

Cognitive Resources Assumption: *Cognitive resources include separate procedural and declarative resources, each of which can independently become a source of processing interference.*

The majority of theories of dual-task performance treat central cognition as a single resource, focusing on the ability to translate a perceptual stimulus into a motor response (e.g., Pashler, 1994). In contrast, we adopt the view that cognition can better be characterized as two distinct resources (e.g., Anderson et al., 2004; Meyer & Kieras, 1997; c.f. Laird, Newell, & Rosenbloom, 1987): a declarative resource that serves as a storage memory for factual knowledge, and a procedural resource that integrates information and effects new behavior. This distinction facilitates generalization beyond simple stimulus-response selection to a wide range of complex tasks. In addition, the distinction helps to account for a separation between memory-related and procedural interference that sometimes occurs in multitasking, including how practice and learning can lead to proceduralization and changing usage of cognitive resources as described shortly.

Declarative Resource Assumption: *Cognition's declarative resource represents static knowledge as information "chunks" that can be recalled (or forgotten).*

Declarative memory can be characterized as long-term storage of information, and the declarative resource allows for access to this information while accounting for the effects of learning, decay, and retrieval failures. For our computational framework, we adopt the ACT-R (Anderson et al., 2004) representation of this information in terms of "chunks" of knowledge, each of which includes a set of attribute-value pairs such that the chunk's "type" defines its attributes. For example, the chunk

```
Three-Plus-Four
    isa         addition-fact
    addend1     Three
    addend2     Four
    sum         Seven
```

represents the knowledge of the addition fact 3+4=7. The type "addition-fact" defines the three attributes corresponding to the addends and sum, and the attribute values Three, Four, and Seven are themselves chunks; thus, a set of chunks essentially forms a network of declarative knowledge. The declarative module can receive requests to retrieve particular chunks of knowledge on the basis of a partial pattern (e.g., 3+4=?). If it succeeds in retrieving a chunk, it will place it in the retrieval buffer, the buffer associated with the declarative resource. The declarative module can only handle one retrieval request at a time. The chunk representation serves as the base representation for all information passed between production rules and the various resource modules through the buffers — that is, resource processing requests as well as processing results are both represented by chunks in the associated module's buffers.

In addition to the representation of chunks, ACT-R defines several equations that govern the probability of chunk retrieval (versus forgetting) and the latency of retrieval, both of which are dependent on chunk activation that decreases over time but increases with practice and use. The specifics of the memory theory are relevant to threaded cognition insofar as they predict the latency of retrievals of a given memory chunk; we will provide details as necessary in the context of the model simulations described later. For now, it suffices to state that the declarative resource can be considered a source of cognitive interference independent of the procedural resource, particularly in situations in which processes require frequent and potentially competing retrievals from declarative memory.

Perceptual and Motor Resources Assumption: *The perceptual and motor resources allow for information acquisition from the environment and action in the environment.*

As mentioned earlier, the human system incorporates a variety of perceptual and motor resources to acquire information and act in the external world (respectively). Of the many possible resources, our treatment focuses on the perceptual and motor resources most central to the types of tasks we wish to model: the visual and auditory perceptual resources, and the manual motor resource. Threaded cognition relies on the computational formulation of perceptual and motor resources in ACT-R, parts of which (particularly the motor) are derived from the EPIC cognitive architecture (Meyer & Kieras, 1997). The visual resource has both a "where system" and a "what system": the where system produces preattentive visual processing that finds objects in the environment based on spatial location and visual properties, and the what system identifies the object and places a declarative representation of the object in the visual buffer. An extension of this visual resource (Salvucci, 2001-b), derived from a computational model of eye movements in reading (Reichle, Pollatsek, Fisher, & Rayner, 1998), predicts the observable movements of the eye that correspond to the unobservable shifts of visual attention. The auditory resource has analogous where and what systems for audible stimuli, and places its results in the aural buffer. The motor resource assumes a two-hand manual module centered on standard interaction movements on a desktop computer (e.g., typing and mouse movement). The timing and other parameters of vision, audition, and motor movement are detailed in Anderson et al. (2004), along with their original treatment in Meyer & Kieras (1997).

Procedural Resource Assumption: *Cognition's procedural resource represents procedural skill as goal-directed production rules.*

The procedural resource, the central resource in our view of threaded cognition, integrates and maps currently available results of resource processing into new requests for further resource processing — just as the cook in our earlier analogy transfers the end results of cooking processes (e.g., mixed batter) to initiate new processes (e.g., baking the batter into a cake). For example, in the context of a simple choice task, the procedural resource may map an encoded visual stimulus and a recalled associated response into the request to perform a motor command. Mappings may be characterized as and implemented by a wide variety of theoretical and computational constructs. For instance, one could conceptualize the mapping in terms of schemata (Norman & Shallice, 1986; Rumelhart, 1980; Rumelhart, Smolensky, McClelland, & Hinton, 1986) or dynamic gating mechanisms (O'Reilly & Frank, 2006).

We adopt the representation of such mappings as condition-action production rules, currently in use as the core component of several cognitive architectures (e.g., EPIC: Meyer & Kieras, 1997; Soar: Laird, Newell, & Rosenbloom, 1987; 4CAPS: Just, Carpenter, & Varma, 1999). A production rule (which we will often refer to as simply a "rule") defines a set of conditions and actions, such that the conditions must be met for the rule to execute (or "fire") the given actions. In the ACT-R formulation of a production rule, both the conditions and actions utilize buffers for information transfer: the conditions collate and test information placed in the buffers by their respective modules, and if the rule fires, the actions place new requests for resource processing in the buffers. In addition to the buffers provided by the various other resources, the system has a "goal buffer." It can be considered as the procedural resource's own buffer, which stores information about the current goal of the system. Typically, production rules include a condition for the goal buffer that matches only for goals of a particular type — for instance, a rule that concerns a choice task matches and fires only when the current goal indicates that the system is attempting to perform a choice task. The goal buffer thus directs the production system only to rules currently relevant to the task at hand. (This goal-directed aspect of a production system relates to similar paradigms in alternative frameworks, such as the activation of relevant action schemas and inhibition of irrelevant ones in contention scheduling: Norman & Shallice, 1986; Cooper & Shallice, 2000).

In addition to matching and adding contents to the buffers, we assume that a production rule can test the status of the modules and buffers in two ways. First, a rule can test whether a module is busy or free — that is, whether or not the module is currently performing a resource-processing task, such as visually encoding a stimulus or executing a motor movement. Second, a rule can test whether a module's buffer is currently empty or filled: A module fills a buffer with the results of information processing when completed, and the buffer empties when a rule matches and uses this information; for example, when visual encoding completes, the visual module places the encoded information in the visual buffer, and then the buffer empties when a subsequent rule uses this information. These two aspects of the modules and buffers are critical to our treatment of threaded cognition in that production rules can discriminate when a particular resource is in use: if the module is busy (processing) or the buffer is full (storing results of processing), the resource can be considered in use; otherwise, it can be considered free, and thus a rule can initiate processing on the resource if desired.

As an example, consider a scenario in which a person performs a simple visual-manual choice task that involves visually encoding a stimulus and pressing an associated response key. Let us also assume that when the visual stimulus is presented on the left — for instance, an "O" on the left side to produce the stimulus "O – –" (Schumacher et al., 2001) — the person should respond by pressing the right index finger. Such a task could be implemented with two production rules:

```
Attend-stimulus
IF      the goal buffer contains a choice task
        and the visual resource is free and the visual buffer is empty
THEN    issue a request to the visual resource to encode the stimulus

Respond-to-left-stimulus
IF      the goal buffer contains a choice task
        and the visual buffer contains the stimulus “O – –”
        and the manual resource is free
THEN    issue a request to the manual resource to press the index finger
```

Both rules are directed by the same goal, namely the goal to perform the choice task, and this goal is matched against the current goal in the goal buffer. The first rule tests the status of the visual resource and issues a request to encode the stimulus (assuming for simplicity that the location of the visual stimulus is known). When the visual resource processing finishes, the second rule uses the information in the visual buffer (which also clears the buffer) and generates a motor request.

We can visualize the execution of this model in two ways. Figure 2(a) shows a process timeline illustrating the steps of the choice task and the resources involved in each step, where time flows from top to bottom, and the height of the box represents the time needed to execute that activity. As is apparent in this view, the process alternates between rule firings in the procedural resource and processing in the peripheral (visual and manual) resources. Each rule firing requires 50 ms on the procedural resource, an estimate that has been derived over several decades of experimentation and is shared by the major architectures (including EPIC and Soar). It is important to note that while this central procedural resource is needed at each step to initiate new resource processing, the procedural resource is not continually busy but rather has periods of waiting for resource processes to complete (just as the cook waits for food to finish baking, frying, etc. before proceeding); this aspect of the system will be critical in our upcoming specification of how threaded cognition interleaves the procedural processing for multiple tasks.

<< Insert Figure 2 approximately here >>

Figure 2(b) shows a storyboard view illustrating how information passes between resources. The figure shows an overview of the resources including processing modules (rounded rectangles) and buffers (squared rectangles), with the procedural resource at the center. (Buffers for the manual and vocal resources are not shown for simplicity, given that these resources do not produce results to be used by the procedural resource.) Each individual storyboard diagram tracks the firing of a production rule, where arrows from the buffers to the procedural module indicate rule conditions, and arrows pointing back to the modules indicate rule actions. The detection of the stimulus by the visual resource, combined with the goal in the goal buffer, trigger the first rule that initiates the encoding process in the visual module. When encoding is completed, the resulting representation (i.e., the encoded stimulus) is placed in the visual buffer. The representation in the visual buffer, again combined with the current goal, triggers the second rule that initiates the appropriate motor response. In both cases, the goal buffer guides the model by matching along with the other conditions, providing top-down control of model processing along with the more bottom-up triggering that arises from stimulus detection and encoding.

Procedural Learning Assumption: *When learning new tasks, declarative task instructions are gradually transformed into procedural rules that perform the task.*

A critical aspect of the dual cognitive resources lies in the gradual transformation of declarative task representations into procedural representations. In particular, we adopt the theory of *production compilation* (Taatgen & Lee, 2003) that provides a computational mechanism by which task instructions encoded as chunks change into ACT-R production rules with practice over time. First, task instructions are encoded into declarative memory, typically by reading (visual encoding) of phrases and/or sentences that specify these instructions. (In the absence of a full-fledged syntactic parser, the instructions are often presented in simple albeit still realistic language.)

Once instructions are stored as declarative knowledge, a general set of interpreter production rules retrieves each instruction and executes its desired actions. As these interpreter rules execute the declarative instructions, the production compilation mechanism begins to combine the general interpreter rules and the instructions into task-specific production rules. Production compilation is a slow process in the sense that a new rule must typically be re-learned multiple times before it is able to compete successfully with existing rules.

Clearly this learning process has significant implications for resource usage and thus for multitasking in general. Initially, while task knowledge primarily lies in declarative memory, both the declarative and procedural resources experience heavy use — the declarative for repeated retrieval of instruction chunks, and the procedural for repeated interpretation of these chunks. Later, after compilation of task knowledge into production rules, declarative resource use drops significantly or entirely, leaving only the procedural resource to perform the task. This process explains why multiple new tasks are hard to combine: not because an executive process can only attend to one new task at a time, but because two new tasks both heavily depend on declarative memory as a resource that they almost continuously need.

Core Assumptions for Threaded Cognition and Multitasking Performance

Building on the single-task assumptions above, threaded cognition adds several critical assumptions that describe the nature of multitasking performance. While some of the theoretical aspects of these assumptions have been explored in other contexts (e.g., Kieras et al., 2000), threaded cognition provides a novel integration of these ideas in order to formulate a domain-independent, general multitasking model. Threaded cognition also extends the existing computational framework of ACT-R, resulting in a new "threaded" version[1] of the architecture that greatly facilitates the representation and modeling of multitasking behavior. In fact, the

[1] The code that implements threaded cognition and all models described in this paper are publicly available at < http://www.cs.drexel.edu/~salvucci/threads >.

threaded architecture allows for immediate integration of two or more single-task models into a multitasking model and immediate prediction of the resulting multitasking behavior.

Threaded Processing Assumption: *Cognition maintains a set of active goals that produce "threads" of goal-related processing across available resources.*

When doing a single task, system resources — and the procedural resource in particular — execute in the service of a single goal. The goal, as mentioned, directs procedural processing by focusing attention on task-relevant rules and away from task-irrelevant rules. To generalize this view, this first assumption of threaded cognition simply posits that cognition maintains a set of active goals — that is, a set of goals that the system is currently trying to satisfy at the same time through concurrent multitasking. Of course, the primary challenge that arises is how the system then allocates processing and balances execution among resources in order to run all goals to completion, and it is this challenge that will be addressed in subsequent assumptions.

In terms of our computational formulation, the current ACT-R architecture (Anderson et al., 2004) allows for a single goal, represented as an ACT-R declarative chunk, to reside in the goal buffer and direct rule firings. We extend this formulation to allow multiple goal chunks to co-exist in the goal buffer, and in addition, we generalize the rule specification mechanism to allow for the creation of multiple goals. In the current ACT-R theory, when a rule specifies a new goal, this new goal replaces the current goal in the goal buffer. For example, the rule

```
IF      the goal buffer contains a choice task
        and the motor response has been issued
THEN    set a new goal to wait for a new stimulus
```

represents the termination of the current choice-task goal and, simultaneously, the initiation of a new waiting goal, since this latter goal replaces the former in the goal buffer. Our new framework allows for multiple goals to be created in the action side of a rule; for instance, to direct the model to perform both the choice task and a secondary task, a rule could be specified as follows:

```
IF      the goal buffer contains a dual task
THEN    add a goal to perform the choice task
        and add a goal to perform the secondary task
```

Both goals are thus created and added to the set of active goals maintained by threaded cognition; in other words, the goals are both added to and maintained in the new multi-chunk goal buffer. In addition, a rule can indicate completion of the current goal by simply removing the current goal from the set:

```
IF      the goal buffer contains a choice task
        and the motor response has been issued
THEN    remove (terminate) the current goal (i.e., the choice-task goal)
```

Each goal in the active goal set activates relevant rules and, in executing these rules, produces a stream of processing across the system resources. We define a *thread* as all processing in service of a particular goal, including procedural processing through the firing of rules and other resource processing initiated by these rule firings. For example, Figure 3(a) illustrates two threads for distinct, concurrent choice tasks: the visual-manual choice task seen earlier, and an aural-manual choice task in which the stimulus is aural rather than visual. After a 50 ms delay in detecting a tone (details forthcoming in our later description of the dual-choice domain), this thread fires a rule to encode the tone (i.e., determine whether the tone is high or low), and a second rule to generate the proper response. In this case, the aural-manual thread must wait for the manual resource to become free, thus causing a delay in overall reaction time for this task. Figure 3(b) illustrates a visual-manual combined with an aural-vocal thread, thus eliminating conflicts for perceptual and motor resources. However, the second thread still experiences a delay due to a conflict for the procedural resource, because the visual and aural encoding for the respective threads complete at roughly the same time. Figure 3(c) illustrates the same two tasks with less time needed for visual encoding; we see here that no conflicts arise whatsoever, and thus perfect time sharing is achieved. As another view of resource processing, Figure 3(d) provides the storyboard view of this final example. At this point in our exposition, we have not refined our assumptions enough to clearly define how the two threads would be executed on the resources. The subsequent assumptions will further refine this specification to define exactly how multiple threads execute on the processing resources.

<< Insert Figure 3 approximately here >>

Conceptually, a thread represents a particular stream of thought that is associated with some currently active task. A simple task, such as one of the choice tasks illustrated above, likely has only one associated thread. However, many complex tasks could be represented as a set of threads that collectively work toward a higher-level goal but each individually focuses on a particular aspect of that goal. Very complex dynamic tasks, such as driving or air-traffic control, are easily thought of as incorporating multiple task threads. For instance, driving and air-traffic control involve both monitoring of the environment for situation awareness and action in the environment to alter its state; while these threads are clearly related in serving the high-level goal of safe control (e.g., performed actions are often related to a monitored event), they can also act independently and communicate information when needed. However, tasks need not be this complex to be amenable to a threaded approach. For example, models of list-memory typically incorporate task processes of both stimulus encoding and list rehearsal (e.g., Anderson, Bothell, Lebiere, & Matessa, 1998) that could be viewed as distinct threads in service of the larger task goal. As another example, models of transcription typing (e.g., John, 1996) include, in essence, separate threads to read text and simultaneously type this text.

A threaded perspective on such tasks allows for a straightforward account of the kinds of flexibility exhibited in human multitasking. In particular, independent threads are not inextricably tied to one another in a processing sense, but rather are combined or separated however needed for maximum flexibility. For instance, in the transcription-typing example above, both the reading and typing threads represent well-practiced skills that can easily operate independently of the other. In the driving or air-traffic control example, it is sometimes the case that another person can assist with the monitoring task (e.g., a team of air-traffic controllers with assigned tasks, together managing a particularly busy airspace, or a vehicle passenger who checks for oncoming traffic); in this case, the original person can eliminate their own monitoring thread and concentrate their efforts on performing necessary actions. A threaded perspective of such tasks provides an account for how people integrate their skills as necessary to adapt to the demands of the current task environment.

Resource Seriality Assumption: *All resources — cognitive, perceptual, and motor — execute processing requests serially, one request at a time.*

One of the most important core assumptions of threaded cognition is that all resources operate sequentially, processing only one request (and thus serving only one thread) at a time. At first glance, this assumption may seem quite surprising, running counter to the well-known degrees of parallelism in the brain's neural circuitry. We should make clear that while our characterization of human resources and their associated processes certainly represents an approximation of the true system, we would argue that it provides a *useful* approximation in providing those distinctions most critical to understanding multitasking at the level of abstraction addressed in this paper. In particular, our theory allows for the inclusion of parallelism at the level of multiple resources (e.g., parallel visual and aural processing) and parallelism within each resource (e.g., visual processing of scenes or parallel activation competition for declarative retrieval), but requires sequential processing at the level of an individual resource — a separation which, for the types of tasks addressed in this paper, nicely captures both the power and limitations of multitasking behavior.

To account for limitations on multitasking performance, theories have typically constrained resources in one of two approaches. One approach (e.g., Just, Carpenter, & Varma, 1999; Wickens, 2002) places capacity limits on resources such that each process occupies some percentage of the total capacity, and that multiple processes can execute on the resource as long as the total usage across processes does not exceed the total capacity. A stricter approach (e.g., Anderson et al., 2004) posits that resources can only execute one requested process at a time, forcing subsequent requests to wait until completion of the current process. We prefer the latter "exclusive-use" approach for two reasons. First, the exclusive-use approach represents a stronger, more limiting theory in that the capacity approach can mimic the exclusive-use approach but not vice-versa; the capacity approach also requires additional parameters that define both the total capacity for a resource and the capacity usage for any given process, a large space of new parameters. We start with the stronger theory in order to evaluate its potential for accounting for behavior in common task domains. Second, the exclusive-use approach generalizes in a straightforward way across all cognitive, perceptual, and motor resources. In contrast, in a parallel capacity-limited approach, the manner in which different resources utilize parallelism would depend on the particulars of that resource, notably in terms of whether the resource contains a serial bottleneck — for instance, any parallelism in the motor system must eventually be resolved to direct a single physical resource (e.g., moving the finger to a location), and thus such resources must resolve conflicts and serialize processes somehow within the system. We thus prefer the unified exclusive-use approach, and for the sake of parsimony, require that all resources abide by this approach (in contrast to, e.g., EPIC's parallel cognitive processor but sequential motor system: Meyer & Kieras, 1997).

The most important implication of the resource seriality assumption arises in the procedural resource, where the assumption forces the requirement that only one rule proceed at a given time. In general, as a production system matches and fires rules, situations may arise in which multiple rules could potentially fire at the same time (i.e., the conditions for multiple rules match the current state of the buffers). Some cognitive architectures such as EPIC (Meyer & Kieras, 1997) allow for multiple rules to fire simultaneously in these situations. In contrast, threaded cognition follows the spirit of the ACT-R cognitive architecture and requires that rules fire sequentially. Anderson et al. (2004) provide a neural argument for the single-rule requirement, namely that in cases where the current goal activates (i.e., matches) more than one rule, inhibitory projections from the striatum to the pallidum allow a single rule to dominate over the others and produce its selected action (see Graybiel & Kimura, 1995). In a sense, threaded cognition generalizes ACT-R's mechanisms to apply across a set of active rules — that is, when multiple rules are activated by currently active goals, this same mechanism produces only a single dominant rule and only this dominant rule is allowed to proceed.

Procedural seriality is critical to the workings of threaded cognition because it posits the existence of a procedural bottleneck separate from potential bottlenecks in other resources. For example, returning to the dual-choice example illustrated in Figure 3(b), the visual-manual and aural-vocal choice tasks utilize different perceptual and motor resources, and thus there is presumably no potential for perceptual or motor bottlenecks in the dual-task case. However, as the figure illustrates, there may still be a bottleneck or conflict for the procedural resource. The procedural bottleneck is essentially the "central bottleneck" that some empirical studies have noted in dual-task performance (see, e.g., Pashler, 1994). Put in other terms, one could view procedural seriality as stating that cognition can only process resource results and issue resource requests for one thread at a time; nevertheless, one thread's resource processing can proceed in parallel with another as long as they do not both require procedural processing at the same time.

While a serial procedural resource runs counter to some notable cognitive architectures, there is general agreement that other system resources — perceptual, motor, and declarative resources — can be well characterized as operating in a sequential manner (though process execution may contain parallelism, such as visual processing of an external scene). Our incorporation of ACT-R's perceptual and motor resources (Anderson et al., 2004; see also Byrne & Anderson, 2001), which as mentioned largely derive from those of EPIC (Meyer & Kieras, 1997), follow the assumptions of these architectures that these resources require serial processing. ACT-R's declarative resource, which handles retrievals from declarative memory, is similarly assumed to require serial processing.

The resource seriality assumption generates immediate predictions about resource conflicts in the case of multiple threads. Returning to the example in Figure 3(a), the two illustrated tasks share the manual resource as the necessary response modality. In contrast to the short delay caused by the procedural conflict in Figure 3(b), the manual resource conflict forces the aural-manual thread to wait until completion of the visual-manual thread's motor response, creating a long delay in processing. This type of resource contention can, and typically does, produce more significant delays in the component tasks than contention for the procedural resource, given that perceptual and motor processes can take significantly longer than the 50 ms rule-firing time for the procedural resource.

The bottlenecks that arise from resource conflicts are often associated with perceptual and motor resources, but they need not be; in fact, one of the most interesting examples of resource contention arises in the declarative resource, particularly in cases that involve practice and learning. Earlier we discussed the mechanism of production compilation, which transforms declarative instructions into procedural rules that perform a task. This mechanism predicts that declarative memory retrievals are frequent in the early stages of learning, but gradually drop out as the instructions evolve into procedural rules. Generalizing this to the case of multiple threads, threaded cognition may result in diminishing interference over time for two reasons: the learning thread gradually uses the declarative resource less frequently and avoids conflict with other threads that require this resource, and the learning thread fires fewer production rules overall and thus requires less processing on the procedural resource.

Resource Usage Assumption: *Threads acquire and release resources in a greedy, polite manner.*

Further specification of how threaded cognition manages resource processing requires some description of how the threads themselves acquire and release resources. Using Kieras et al.'s (2000) terminology, threaded cognition can be characterized as providing a "liberal" managerial style that allocates resources in a "tolerant, laissez-faire" manner, where threads largely manage their own usage in requesting and freeing resources — as opposed to a "conservative" executive that allocates in a "strict regimented style." Such a managerial style, however, could be abused by processes that unfairly monopolize resources, and thus creates a need for a "process etiquette" within the threads.

Our formulation of threaded cognition defines such an etiquette by requiring that thread usage of available resources be "greedy" and "polite." Threads acquire resources in a "greedy" manner by requesting resources as soon as possible when needed. (Note that all resource requests occur in the actions of the rule currently firing, and the firing of this action coincides with the acquisition of the resource.) In contrast, a few related efforts have explored how to model multitasking that involves more complex, non-greedy solutions to resource contention. For example, Freed (1998) has described a "sketchy planner" called APEX that has the ability to explicitly delay, interrupt, and prioritize multiple task threads. Norman and Shallice's (1986) supervisory attentional system allows, in principle, for schemas that direct the same kinds of non-greedy planning and prioritization. Howes et al. (2004, 2007) have described a "cognitive constraint modeling" methodology that enables exploration of strategic differences in multitasking behavior through specification of behavioral constraints. We believe that these more complex multitasking behaviors may indeed be exhibited in some circumstances, and threaded cognition in principle does allow for complex multitasking through the presence of additional supervisory threads. However, we also believe that the greedy approach to resource contention is the more common one, especially at the temporal grain size focused on here, and as we demonstrate in the later model simulations of a variety of tasks.

After acquiring a resource, a thread releases resources in a "polite" manner by releasing, or freeing, a resource for other threads as soon as its processing is no longer required. As mentioned, for threaded cognition and its computational instantiation in ACT-R, resources are in use when (1) they are currently performing a processing request in service of some thread, or (2) the results of a processing request (if any) remain in the resource's buffer still unused by the requesting thread. Rules that request a particular resource thus must check that the resource is not busy and that its associated buffer is empty. After the thread has acquired the resource, the requested process either terminates with no results (for a motor request) or places a result in the buffer (for a perceptual or declarative request); in the latter case, a subsequent rule collects the result and thus clears the buffer. The resource, no longer busy and with an empty buffer, can immediately accept new requests from other threads. Thus, politeness is built into the system in that typical threads cannot monopolize resources because they require the results of the requested processes. (Nonetheless, a purposefully impolite thread could, in theory, request a result and never use the result, leaving it in the buffer and maintaining control of the resource; in practice, however, such behavior would not evolve naturally through the production compilation process, and could also be avoided through the use of an appropriate decay mechanism.)

Figure 3 illustrates the greedy, polite threads that comprise the dual-choice task. In each of these figures, both threads greedily acquire resources when ready. For instance, in Figure 3(b), the visual-manual thread immediately encodes the stimulus and, upon completion, immediately issues the response; the aural-vocal thread does not attempt to (nor can it) schedule or plan around the other threads' resource acquisitions, but rather simply acquires resources when available and waits when they are not. In Figure 3(a), where the visual-manual and aural-manual threads contend for the manual resource, the completion of manual resource processing immediately frees the resource, allowing the aural-manual rule that issues the motor response to proceed on the procedural resource.

Conflict Resolution Assumption: *When multiple threads contend for the procedural resource, the least recently processed thread is allowed to proceed.*

Typically, when a thread desires a particular resource and that resource is currently busy, that thread simply waits until completion of the current process and then acquires that resource, as illustrated in our earlier examples. In certain cases, however, when that resource becomes free, there may be two or more threads waiting to acquire the resource (possibly including the thread that just used and released the resource). We thus require some way to specify, in these cases, which thread may proceed and acquire the resource.

Deciding which thread may acquire a resource reduces to the problem of deciding which thread may fire a rule on the procedural resource, since resource acquisition can only occur through a rule firing. To make this decision, threaded cognition adopts a policy that the least recently processed thread — that is, the thread that has least recently fired a rule on the procedural resource — is allowed to proceed. The primary motivation for the least-recently-processed (LRP) policy is that it provides a parsimonious way to balance processing among threads: by ensuring that threads have a regular opportunity to progress through the firing of procedural rules, the system allows all threads a chance to acquire resources and avoids starving any thread of processing time. When two threads exhibit similar resource usage, the LRP policy results in an alternation of rule firings — the procedural resource can fire a rule for one thread while another thread's peripheral processes (vision, motor, etc.) are running, then vice-versa, and so on, achieving highly efficient parallelism between the two threads. Our upcoming model simulations of dual-task behavior, including those in which people exhibit perfect time sharing, demonstrate how such processing arises from threaded cognition. When two threads exhibit very different resource usage, such as when one task includes lengthy peripheral processes (e.g., motor movements) whereas the other requires frequent procedural steps, the LRP policy allows for the high-frequency task to execute at high efficiency but still allows the low-frequency task to acquire the procedural resource when attention is needed.

Relation to Other Theories of Multitasking

Threaded cognition builds on a number of existing ideas and theories and attempts to unify them under the auspices of a single computational theory. The earliest descriptions of single-channel bottlenecks (e.g., Craik, 1948; Telford, 1931; Welford, 1952) could be characterized within our theory if one considers the entire (short discrete) task, and all the resources involved (i.e., cognition plus perceptual and motor resources), as a single processing channel. Later theories that emphasize particular resources, such as bottlenecks that arise due to stimulus perception (e.g., Broadbent, 1958) or response selection (e.g., Pashler, 1994), also correspond nicely with aspects of the theory and its integration in the cognitive architecture; for example, the visual perception and eye-movement aspects of the theory govern visual stimulus perception, while the declarative-memory aspects of the theory, including recall and forgetting of memory chunks, govern at least some components of response selection.

The most closely related theories to our own are those that attempt to integrate a number of modalities and resource constraints into a unified framework. One well-known theory is the multiple resource theory of Wickens (2002; see also 1984), which characterizes dual-task interference in terms of four dimensions: stages (perceptual/cognitive versus response), sensory modalities (auditory versus visual), codes (visual versus spatial), and visual channels (focal versus ambient). The theory incorporates a methodology by which one can create a numerical "conflict matrix" that quantifies the resource conflicts between tasks across dimensions, and allows for subsequent computation of the amount of "total interference" (as a single numeric value). While multiple resource theory and its associated modeling methodology has the potential to capture multitasking behavior and interference at a high level, it does not provide some of the many advantages of the more detailed modeling possible in a cognitive architecture — for example, prediction of the fine-grained temporal sequence of behavioral events and prediction of actual quantitative measures to be compared with human behavior (e.g., measures ranging from reaction time to keystroke sequences to driver performance characteristics, as seen in the upcoming model simulations).

A related effort arises in Cooper and Shallice's (2000) computational implementation of contention scheduling (Norman & Shallice, 1986). Contention scheduling allows for triggering of appropriate actions through both top-down control and environmental stimuli — an approach very similar to our own — and this action-selection mechanism can be the primary bottleneck in dual-task performance. The computational model of contention scheduling further fleshes out the original theory, with a demonstrated application in the real-world domain of coffee preparation. At the same time, the model and its application focus on sequential action selection in this domain rather than multiple concurrent activities. In principle, this approach allows for multitasking if there are no resource conflicts between tasks, including both cognitive and effector (perceptual and motor) resources. However, in the presence of resource conflicts, the theory relies on higher-level schemas in the supervisory attentional system to resolve the conflicts, and at this time there is no theory of the learning and behavior of such schemas. Thus, the contention scheduling model at present does not fulfill our desire of a general model of multitasking performance for arbitrary tasks.

A separate effort by Liu, Feyen, and Tsimhoni (2005) explores the representation of cognitive models as queuing networks within their QN-MHP architecture (see also Liu, 1996). The queuing networks provide a mathematical framework with which to express a parallel network of servers, each of which may process information serially or in parallel. As such, the architecture provides a computational mechanism for concurrent task execution without the need for task-specific executive processes. At the same time, the architecture employs "separate servers to handle distinct goal execution functions such as goal prioritization, complex cognitive processing …, performance monitoring, and procedure selection" (Liu et al., 2005). In contrast, threaded cognition does not require specialized modules for task priorities or procedure selection, relying solely on its threading mechanism to execute concurrent tasks. Also, the queuing networks leave open the question of serial versus parallel processing at the server level; threaded cognition is the narrower and stronger theory in its commitment to resource seriality where resources process a single request at a time.

The treatment of multitasking most closely related to our own is that of Kieras et al. (2000), who studied production-system models of multitasking in the context of the EPIC cognitive architecture (Meyer & Kieras, 1997). In particular, they explored potential characteristics of "general executive" processes that govern across task domains, in contrast with "customized executive" processes that define domain-specific aspects of multitasking performance. In some ways, one could characterize threaded cognition as, in their terminology, a "liberal general executive with polite task processes": task models are "polite" in the sense that they request resources only when needed and release resources when done, allowing threaded cognition to be "liberal" in allowing task processes to proceed whenever they can. However, threaded cognition differs from the Kieras et al. conceptualization in that it does not utilize rule-based executive processes to manage resources and resolve resource conflicts, but rather posits a parsimonious architectural mechanism that performs this work for all tasks. In a sense, threaded cognition introduces the function of a general executive without the need for an explicit executive system or process. Another important difference is that the EPIC architecture allows multiple rule firings in parallel in contrast to our inclusion of a serial procedural resource. Nevertheless, our work follows very much in the spirit of Kieras et al.'s (2000) effort, and we have examined and modeled some of the same tasks (described in the section on model simulations) to further compare and contrast our approaches.

Threaded cognition arose from a unification of two recent modeling approaches by the individual authors. Salvucci (2005) explored an ACT-R general executive that incorporated queuing and timing mechanisms to achieve multitasking behavior. This approach led to more conservative multitasking because, unlike threaded cognition, only one goal was active at a time, which led to cases in which other goals could not proceed even though the currently active goal was waiting for a peripheral resource. This work also required a new architectural extension for timing goals on a queue, whereas threaded cognition requires no such extension. Taatgen (2005) modeled dual-task performance such that, in essence, both task goals were embedded into a single representation. While this approach achieved more efficient multitasking than that of Salvucci (2005), it relied on more tightly coupled task representations that made it more difficult both to learn tasks individually and to perform tasks independently when desired. In addition, there were no constraints on when each task could proceed (in contrast with threaded cognition's preference for the least recently processed thread), and thus nothing prevented unbalanced processing or starvation of one task over another. We view threaded cognition as incorporating the best of each approach — a straightforward, parsimonious theory that accounts for the highly efficient yet highly flexible nature of multitasking observed in human behavior.

Key Claims and Predictions of Threaded Cognition

We can summarize the key claims of threaded cognition theory as follows:

1. Cognition can maintain and execute multiple active goals, resulting in concurrent threads of resource processing.
2. Threads can be characterized as alternating blocks of procedural processing (i.e., rule firings that collect information and initiate new resource requests) and processing on peripheral resources (including perceptual, motor, and declarative memory resources).
3. Processing interference can arise on the central procedural resource as well as on the declarative, perceptual, and motor resources.
4. Threads acquire resources greedily and release resources politely, which arises naturally from the characterization of resources as modules and buffers.
5. Cognition balances thread execution by favoring least recently processed threads on the procedural resource.
6. With practice, threads become less dependent on retrieval of declarative instructions, reducing conflicts for both the declarative and procedural resources.
7. Cognition requires no central or supervisory executive processes; instead, multitasking emerges from the interaction of autonomous process threads in conjunction with the key claims above.

These claims give rise to several general, qualitative predictions about the nature of dual-task performance. When two tasks require common perceptual or motor resources, dual-task performance for one or both tasks will typically be impaired. Even when the tasks utilize distinct perceptual or motor resources, dual-task performance may be impaired due to potential procedural bottlenecks. However, in the absence of procedural interference, dual-task performance can be equivalent to single-task performance. In addition, with practice of either or both tasks, dual-task interference will typically lessen due to the gradual compilation of declarative instructions into procedural form. These predictions have all been supported by empirical work examining the central bottleneck in dual choice tasks with distinct modalities (see Pashler, 1994), perfect time sharing in dual-task performance (e.g., Hazeltine, Teague, & Ivry, 2002; Schumacher et al., 1999), and practice and learning effects in dual-task behavior (Schumacher et al., 2001).

At the same time, the computational realization of threaded cognition provides much more than general qualitative predictions: it immediately predicts the behavioral process that results from performing two or more tasks concurrently. One could begin with two or more models of individual tasks, each independently developed as a sound model of behavior for the respective domain. Given such models, threaded cognition allows for simulations in which the model execute concurrently, generating predictions about multitasking performance. In addition, such models typically interact with a simulated environment that collects data and analyzes behavior for many relevant observable measures, moving beyond simple reaction time measures. Thus, the space of testable predictions for threaded cognition is enormous: it claims that for any two or more independently validated task models, the combined multitasking model as produced by threaded cognition will match the behavior of people performing these same tasks concurrently. In the next section, we sample from this space of testable predictions and compare the theory's predictions to empirical data for a representative set of task domains.

Model Simulations of Representative Tasks

With the goal of testing our theory of threaded cognition, we chose four representative task domains in which to examine and compare the theory's predictions to human performance: dual choice, tracking and choice, reading and dictation, and driving. These four domains were chosen specifically to illustrate the generality of the theory with respect to four conceptual axes, as outlined in Table 1. First, the domains illustrate how concurrent multitasking sometimes produces interference effects (i.e., degraded performance) in one or both tasks, but sometimes produces no such interference; in fact, two of the domains (dual choice and driving) demonstrate how interference can either occur or not occur depending on specific task conditions. Second, the domains illustrate how practice can change the resulting effects of multitasking behavior over time. Third, the domains span both classic, well-studied laboratory tasks as well as realistic complex tasks, ensuring that the theory generalizes to real-world task domains. Fourth, the domains illustrate how models can be re-used for predictive purposes such that a model of a particular task can be integrated with new tasks to produce new predictions, an important aspect of our integrated theory. We now describe each of the modeling efforts in detail, which all together demonstrate how threaded cognition can account for a broad range of multitasking behavior.

<< Insert Table 1 approximately here >>

Dual Choice: The Perceptual Refractory Period and Perfect Time Sharing

Dual-choice tasks represent the most widely studied task domain in the concurrent multitasking literature. Most dual-choice studies employ the perceptual refractory period (PRP) paradigm, in which one task is given priority over the other. Schumacher et al. (2001) have argued that people cannot fully exhibit their dual-tasking abilities in that paradigm because of the priority between tasks. As a consequence, they argue, people may postpone the secondary task until the primary task is well underway, producing interference effects that can be attributed to the peculiarity of the instructions, instead of being due to cognitive limitations. In order to test this idea, they conducted dual-task experiments in which both tasks had equal priority. In their Experiment 1, participants were given instructions for two choice-reaction tasks. The first task was a visual-manual task, in which a circle would appear on the screen in one of three positions, and a finger corresponding to the position had to be pressed (left position mapped to the index finger, middle position to middle finger, right position to ring finger). The second task was an aural-vocal task, in which a tone of low, intermediate or high pitch was presented, to which participants had to respond “one”, “two”, or “three”, respectively. The tasks were first trained in a single-task condition on day 1, and were then tested in both single-task and dual-task conditions on days 2-5. The results show that participants suffered from dual-task interference on the first days of the experiment, but that this interference disappeared completely by day 5.

The model of this task is based on an earlier version reported by Taatgen (2005), although that model used a specialized goal representation instead of the more general threading approach. At the start of the task the model initiates two threads, one for the visual-manual task and one for the aural-vocal task. We will start examining the model behavior when it is done learning, corresponding to performance on day 5, where people exhibited perfect time sharing — when task performance is equally fast in single-task and dual-task conditions. In fact, we have already used this model in our earlier examples — namely, the model depicted in the process timeline in Figure 3(c) and the storyboard view in Figure 3(d). The only shared resource in this model is the procedural resource, but this does not result in conflicts, because the two task threads only need the procedural resource at times when the other thread is executing other (non-procedural) processes. Therefore, after many trials of practice, there is no interference between the two tasks.

There is interference, however, in the novice stage, as depicted in Figure 4. This figure shows the process timeline in the very initial stages of task performance, when each procedural step requires time for one rule firing to retrieve the next task instruction, time for the memory retrieval itself, and time for a second rule firing that interprets and performs the task instruction. (We omit the storyboard view of this model due to space constraints; however, one can imagine extrapolating the storyboard in Figure 3(d) into extra rule firings and declarative retrievals as illustrated in Figure 4.) The assumption is that people do not have task-specific rules when they enter the experiment, but rather memorize instructions (as done in the experiment) in declarative form, then interpret these instructions as needed during task execution. Another assumption is that people can immediately map screen positions onto fingers, but that mapping tones onto numbers requires a second memory retrieval. The process timeline illustrates how the declarative representation of the task initially creates dual-task interference, primarily because declarative memory becomes a heavily contended resource; the gray areas in Figure 4 indicate periods of time in which one thread must wait until the other thread releases a resource, causing delays in task performance. With practice, the production compilation mechanism described earlier gradually compiles the retrieval of declarative representations into task-specific production rules. This frees up declarative memory as a resource, removes the bottleneck from the model, and eventually produces the interference-free multitasking shown in Figure 3(c). Figure 5 shows the match between model and data, $R^2 = 0.96$, $RMSE = 0.026$.

<< Insert Figure 4 approximately here >>

<< Insert Figure 5 approximately here >>

To support their claim that the PRP effect arises due to instructions, Schumacher et al. (2001) took the participants from Experiment 1 and put them in a PRP paradigm with the same tasks. The main difference was that they were instructed to give priority to the aural-vocal task, instead of the equal priority of the original task. In addition, the visual stimulus was presented 50, 150, 250 or 500 ms after the aural stimulus (a delay period called the "stimulus onset asynchrony," or SOA). As expected, they indeed found the PRP effect — shown in Figure 6 — even though the same participants were perfectly capable of doing both tasks in parallel while they were in Experiment 1. To model this variation of the task, we assumed that instead of initiating the two threads at the beginning of each trial, only the thread of the primary task (aural-vocal) is initiated, and that the thread of the secondary task is only initiated when the procedural portion of the primary task has completed — that is, at the onset of a vocal response. We otherwise assume a model that has been trained on the task in Experiment 1, to mimic the same situation as that experienced by the experiment participants. Figure 6 shows that this slight modification of the model indeed fits the data from Schumacher et al., $R^2 = 0.95$, $RMSE = 0.016$.

<< Insert Figure 6 approximately here >>

In Experiment 1, visual processing was much faster than auditory processing, ensuring that the moments of response selection never coincided. In contrast, Schumacher et al.'s (2001) Experiment 3 increased the difficulty of the visual-manual task to see whether simultaneous response selection for both tasks would lead to interference. Instead of having a congruent mapping from visual positions to finger locations, participants had to reverse the order: now the leftmost visual stimulus mapped onto the rightmost finger, and so on for all fingers. The results, shown in Figure 7, show that this manipulation indeed makes the task more difficult. Moreover, even at day 6, people do not completely succeed in perfect time-sharing. A more detailed analysis by Schumacher showed that some individuals achieve perfect time-sharing by day 6, but others do not. To model these data we took the model of Experiment 1 and changed the instructions for the visual-manual task: Because the stimulus-response mapping is now incongruent, we required the model to retrieve stimulus-response mappings from declarative memory, reflecting the assumption that an incongruent mapping requires conscious effort to resolve. The model fits are included in Figure 7, $R^2 = 0.98$, *RMSE* = 0.020.

<< Insert Figure 7 approximately here >>

Overall, the Schumacher et al. (2001) dual-task paradigm provides an example where the contended resources include declarative memory, and, in Experiment 3, cognitive processing in terms of rule firings. In this case, the effect of declarative memory retrievals can be initially quite substantial, but it also gradually drops out with practice. The effect of cognitive processing is relatively small in comparison, because the opportunity for a resource overlap is very small if the response selection step only takes 50 ms. Hazeltine, Teague, and Ivry (2002) have conducted a further series of experiments that have explored this central response selection bottleneck. Their conclusion was that given enough training participants can achieve almost perfect time sharing, but that there still remains a small residual interference in the order of 10 ms. These effects have subsequently been modeled by Anderson, Taatgen & Byrne (2005) using models very similar to the ones discussed here, again with the exception that they used a control structure that was specifically designed to accommodate the two tasks. These models are also quite amenable to being cast in terms of our proposed threaded approach.

The dual-task models show that, although the procedural resource is the central resource in our theory, declarative memory can also serve as an important source of interference. Several related studies have explored the role of declarative memory in multitasking behavior. Johnston, McCann, and Remington (1995) have demonstrated that there can be two bottlenecks in a choice-reaction task, which they call input attention, associated with initiating visual perception, and central attention, which is associated with response selection. In our view, however, these are both associated with the procedural resource, because a choice-reaction task requires two production rules to make a response. Pashler & Johnston (1998) do not rule out that there is more than one bottleneck, but maintain that a single bottleneck is the most parsimonious account to fit the data. However, they do not consider the role of declarative knowledge as a source of task knowledge to explain novice behavior, but instead consider retrieval from long-term memory to be part of one bottleneck. The basis for this is a study by Carrier and Pashler (1995) that investigated whether memory retrieval is an independent bottleneck. In a dual-task choice paradigm they found that memory retrieval in one task is postponed by response selection in another task, and concluded that memory retrieval is part of a unitary bottleneck. However, in their conceptualization of the task, perception itself initiates the memory retrieval, while in our framework memory retrieval is always initiated by a production rule. The production rule that initiates memory retrieval can only fire after the response-selection production rules has finished, explaining the interference results of Carrier and Pashler (1995).

In another study, Rohrer and Pashler (2003) let participants do a free-recall task and a choice reaction task in parallel. Contrary to earlier studies (e.g., Baddeley, Lewis, Eldridge & Thomson, 1984), they found a substantial degradation in free-recall due to the concurrent task, and concluded that episodic recall is part of the central bottleneck. However, an account with dual bottlenecks of procedural and declarative can give a better explanation. The difference between the Rohrer and Pashler experiment and earlier studies was that their choice-reaction task had arbitrary mappings, making it necessary to retrieve these mappings from memory — therefore both tasks required declarative memory access, producing interference. If the choice-reaction task does not involve memory retrieval, as in the earlier studies, threaded cognition predicts no interference on free recall.

Tracking and Choice: Dual-Task Interference in a Continuous Task

The dual-choice task domain involves two discrete, one-shot tasks that last less than a second and thus provide an interesting but limited window into multitasking behavior. By replacing one of the tasks with a continuous task that requires constant monitoring and response, we can explore the effects of multitasking in the context of an interaction between a discrete and continuous task. One domain that has received some attention in recent literature (e.g., Chong, 1998; Kieras et al., 2000; Lallement & John, 1998) is that of the tracking and choice task: a person tracks a moving cursor and attempts to keep it centered on a target, and occasionally a choice stimulus appears that requires a choice response. Like past efforts, we base our simulations on the empirical results of Martin-Emerson and Wickens (1992), who ran a version of this tracking and choice task. In their experiment, participants tracked the moving target in either an "easy" or "hard" condition; the difficulty of tracking related to the pseudo-random function that perturbed the target's position. In addition, occasionally an arrow would appear on-screen and the participant would respond with a keypress as to whether the arrow pointed left or right. The position of the arrow was manipulated in the experiment such that it appeared at differing degrees of visual angle down from the tracking target — "stimulus offset" — in order to test the difficulty of the choice task as dependent on its distance from the continuous tracking task.

We modeled the tracking task with three simple rules that find the moving cursor, move the cursor to the target, and reiterate this process. We modeled the choice task with a similarly simple model: one rule that attempts to find the arrow, another rule that recognizes when it was not found and iterates, one rule that attends the arrow when found, and another two rules that generate the response for a left or right arrow. The model runs both component task models as individual threads, and because they vie for visual attention, this resource is being shared between the threads. The model also shares the manual resource in that it utilizes its right hand for movement and its left hand for choice response (note that the cognitive architecture currently specifies a single manual module, with both hands as a single shared resource). The task environment used by the model was based on that implemented by Chong (1998) as a duplicate of the original experiment; most importantly, the perturbation functions for the easy and hard conditions were copied exactly to ensure that quantitative comparison of the resulting measures was made on the same scale. Results were compiled from simulation runs with 16 trials for each stimulus offset and for each tracking difficulty condition, with no architectural parameters estimated to produce model fits.

To illustrate the workings of the model, Figure 8 shows a process timeline representing resource processing immediately before and after the presentation of the arrow stimulus. The first block of the tracking thread shows how tracking comprises an iteration of three rules that find the cursor, move it back to the target, and complete the update. When the choice thread sees the arrow stimulus (the third rule firing), it visually encodes the arrow and generates the appropriate key response. During this time, the choice thread occupies the visual and then the manual resources, delaying the tracking movement (which requires the manual resource for movement and the visual resource to guide the movement). This delay creates interference in the tracking thread, impairing tracking performance for a short time and causing a larger deviation of the cursor from the target.

<< Insert Figure 8 approximately here >>

Figure 9 shows two measures of performance: (a) tracking error as a function of stimulus offset, and (b) choice response time as a function of stimulus offset. In (a), we see a large effect of tracking difficulty in the empirical data (solid lines), as well as a slight but steady increase in error as the offset between tracking target and choice stimulus increases. The model results (dashed lines) show the same effects and quantitatively match the data well, R^2=.97, *RMSE*=.68: it exhibits the difficulty effect because at a constant rate of updating the tracking task, the cursor in the hard condition is perturbed by a greater distance; and it exhibits the stimulus offset effect because of the increased time needed to encode a visual object (the choice stimulus) at a farther distance (a prediction of ACT-R's eye-movement model: Salvucci, 2001-b). In (b), the empirical data show an effect of stimulus offset on choice reaction time but no effect of tracking difficulty. The model results, R^2=.74, *RMSE*=.06, duplicate this effect of stimulus offset for the same reasons as for tracking error, but also do not exhibit difficulty effects because difficulty does not affect encoding of the choice stimulus.

<< Insert Figure 9 approximately here >>

These results compare well with results obtained by other researchers working with different modeling frameworks, such as Kieras et al. (2000) with the EPIC cognitive architecture and Chong (1998) with a hybrid EPIC-Soar architecture. Lallement and John (1998) performed an interesting comparison of these models and one of their own, showing that all the models, and the "modeling idioms" on which they are based, produce behavior that corresponds quite well to the empirical data. Our results are comparable to the results of these other models, but our approach has one major advantage over previous models: while the others required explicit specification of when to switch between tasks, resulting in fairly complex behavioral flow diagrams (see, e.g., Kieras et al., 2000), our model is a straightforward integration of two simple component models with no explicit specification of task switching, instead allowing threaded cognition to manage and execute the two tasks.

Reading and Dictation: Perfect Time Sharing in Continuous Tasks

The tracking and choice task above addresses a case in which a continuous task (tracking) exhibits interference when performed concurrently with another task (choice). For some domains, however, studies have shown that people can achieve perfect time-sharing for continuous tasks just as they sometimes achieve it for simpler tasks, as in the dual-choice task domain seen earlier. The classical case is a study of reading and dictation run by Spelke, Hirst and Neisser (1976). In this experiment, two participants went through an 85-day experiment in which they simultaneously had to read stories for comprehension and write down dictated words. Initially, reading speed was affected by the dictation task, but with practice reading speed approached normal levels, as measured by average reading speed without dictation. The dictation speed was not affected by practice.

The Spelke et al. study is interesting because it takes two tasks in which participants are already skilled and combines them. The disadvantage of the study is that the data are noisy (the participants differed considerably in reading speed) and the reporting does not describe detailed behavior on a trial-by-trial basis. More modern studies, such as those by Schumacher et al. (2001) and Hazeltine, Teague and Ivry (2002) addressed earlier, provide better insight into the exact timing of dual-task perfect time sharing through the use of simpler choice tasks. At the same time, such experiments are less representative for real-life multitasking, because the choice tasks are artificial and trained in the context of the experiment. Thus, the Spelke et al. study provides a complementary view of more realistic multitasking, albeit with less detailed timing data.

In the Spelke et al. (1976) experiment, participants had to read stories while taking dictation. Stories varied in length between 700 and 5000 words, and were selected from American, English and translated European writers. On each daily session, participants had to read three stories. While participants read the stories, the experimenter read words drawn from Kucera and Francis (1967), which they had to write down without looking at their handwriting.

After a participant completed writing a word, the experimenter read the next word. Once a participant finished reading a story, they were tested for either comprehension of the story (10 times/week), or recognition of the dictated words (4 times/week). In addition, once per week participants received a control trial in which they only did the reading task. Accuracy on both story comprehension and recognition did not change much with practice, nor did the writing speed, which was constant at about 10 words/minute. The main change in performance was reading speed, as analyzed for the study's two participants, John and Diane. John, whose normal reading speed was 480 words/minute, slowed down to 314 words/minute in the first week, but improved to 451 words/minute in the sixth week. Diane's normal reading speed was 358 words/minute. Her speed in week 1 was 254 words/minute, but this improved to 333 words/minute in week 6.

To model the reading aspect of the task, we used an existing ACT-R model of reading developed by Lewis and Vasishth (2005). Lewis' model builds up a syntactic parse tree of a sentence by attaching new words into this tree as they are read. It has a fairly straightforward pattern of behavior: for each word to be read, the model visually encodes the word, retrieves its lexical item from declarative memory, retrieves a syntactical node in the parse tree from declarative memory to which the word has to be attached, creates a new syntactical node with the word just read, and attaches it to the retrieved syntactical node. Because the visual module can read the next word while the current word is processed, reading and parsing a word takes slightly over 200 ms using Lewis' parameter settings. The model we constructed for taking dictation is similarly straightforward: it aurally encodes the spoken word, retrieves the spelling of the word from declarative memory, then writes the word one letter at a time. Although writing is a highly trained skill, writing without visual guidance is not; the model therefore requires a declarative retrieval for each letter, where the retrieval is assumed to contain the movement pattern needed to write down the letter. With practice, production compilation produces rules for each letter that produce the movement pattern right away. The only estimated parameter in the dictation model was the manual time to write a letter, which was set such that writing speed averaged 10 words/minute.

Figure 10 shows the operation of the model once it has reached the expert stage. The reading model goes through the cycle of steps indicated above, and the dictation model listens to words, and writes out the letters one at a time. The grey areas indicates that there is some interference each time a new word is dictated, but once the word is written out (which takes on average 6 seconds) there is little interference. The novice model, however, needs two production rules and a declarative retrieval in the dictation task for every written letter (instead of the “initiate-movement” production), creating substantially more interference. The result is that the model’s single task reading performance is 298 words/minute. In the dual-task situation this performance slows down to 229 words/minute in week 1, but then speeds up to 270 words/minute by week 6. Figure 11 shows a comparison between the model and the two participants for each of the first six weeks of the experiment, with performance scaled to maximum reading speed. Threaded cognition nicely captures the adaptation over time in the course of the reading and dictation study.

<< Insert Figure 10 approximately here >>

<< Insert Figure 11 approximately here >>

Driving: Transfer and Integration in a Real-World Domain

The three domains modeled to this point all address behavior in the laboratory with the goal of elucidating the finer timing aspects of multitasking behavior; even the reading and dictation task, which incorporates two common real-world skills, combines these skills into a somewhat unnatural, certainly uncommon task domain. To extend our representative set of tasks to include a more complex real-world task, we now examine driving in combination with several possible secondary tasks. One major benefit of the driving domain is that it allows us to apply the theory to a truly real-world task that many people experience on a daily basis. Another benefit is that it allows us to demonstrate the potential for transferring a single model of behavior across multitasking domains, integrating this model with models of other tasks to predict the behavior that results in a dual-task condition.

Driving Alone

The starting point for our treatment of driving is a computational model of driver behavior that has been tested in a separate context. The ACT-R integrated driver model (Salvucci, 2005, 2006) can navigate a typical multilane highway environment, and has been compared against human driver behavior with respect to performance measures for subtasks such as situation monitoring, curve negotiation, and lane changing. In this paper, all tasks involve only the control (steering and acceleration/braking) aspects of the driver model, not the lane changing aspects, and thus we import the core elements of the driver model that implement control. The driver model in essence boils down to four rules that iterate in sequence, each iteration producing one "update" in which steering angle and acceleration/deceleration pedals are potentially adjusted. The first two rules find the "near point" and the "far point" of the current lane (see Salvucci & Gray, 2004); these two points provide information about the nearby and upcoming lane configurations (respectively), subsequently used by the model in its calculations of steering angles. The third rule sends the motor commands to specialized motor modules for steering and pedal movement, and also directs visual attention to encode the object at the far point (i.e., a lead vehicle or a road point). The fourth rule checks for the stability of the vehicle: if the vehicle is not sufficiently stable as defined by its current lateral position and velocity, the process iterates immediately; if the vehicle is sufficiently stable, the process iterates after some delay (defined shortly). These four rules are illustrated in the process timeline in Figure 12.

In addition, the past versions of the driver model were developed in earlier versions of the ACT-R architecture, and thus we imported the ACT-R 5.0 model from Salvucci (2005) and made some implementation-related, non-theoretical changes to the model to work with the newest architecture, ACT-R 6.0. We also re-estimated three parameter values for the current simulation (see Salvucci, 2005, for parameter descriptions): minor adjustments to the steering scaling factor (0.85) and the stability scaling factor (3.0) to account for slight differences in simulation environments, and a change to the accelerator-to-brake movement delay (600 ms) to represent the minimum movement time (Lee et al., 2002) plus 200 ms motor preparation. The final modification to the model relates to its timing aspects during stable control. The original model incorporated a delay such that, in stable situations — that is, when the car is near the lane center with low lateral velocity — the model delayed the next control update for 500 ms; this delay represents a satisficing assumption in the original model that in stable situations, continual control is not required, and this assumption was found to nicely capture behavior. The driver model used here incorporates the same assumption. However, two of the task domains below (driving and choice, and driving and the sentence-span task) used an additional task that required the driver to brake upon onset of a visual stimulus on a lead vehicle; because of this additional task, we assumed a 50% probability of breaking out of the delay period at each potential rule firing, thus representing the fact that the human drivers had reason to pay further attention to the driving task beyond those needed for basic control. In other work, we have explored how appropriate delay times can be learned and can adapt over time (Salvucci, Taatgen, & Kushleyeva, 2006), but for simplicity we do not include these mechanisms in the current modeling effort.

Driving and Choice

Recently Levy, Pashler, and Boer (2006) ran a study that combines a basic choice task with the very common real-world task of driving — in essence, a PRP task in the context of a real-world driving task. Specifically, their task involved following a lead vehicle while occasionally responding to a choice task; the choice task had four conditions, with either a visual or auditory choice stimulus and either a manual or vocal response. In addition, a specified time after the stimulus onset — namely the SOA time described in the dual-choice section — the lead vehicle's brake lights would occasionally light up and require that the driver press the brake pedal. Thus, the task in essence involved three tasks, namely the discrete choice task, the discrete braking task, and the continuous driving (steering) task —lending the task more external validity as a realistic task, and providing an excellent transition to our subsequent simulations involving the driving domain.

Given the driver model, we adapted it to the Levy et al. task as follows. First, we extended the model such that if brake lights appear on the vehicle, it bypasses its default control of the accelerator and taps on the brake pedal instead. Second, we developed a simple model for the choice task, following the same rule pattern as in the previous choice domains: when the stimulus is detected, the model encodes the stimulus (either visual or auditory) and presses a button that corresponds to the stimulus. However, Levy et al. used a slight variant to typical PRP studies in that the stimulus was actually a single or rapidly-repeated double instance of a visual or auditory stimulus: one choice was presented as a single visual flash on the lead vehicle's rear-view window or a single auditory tone, while the second choice was presented as a two rapidly repeated flashes or tones. In addition, for the manual response, the driver was required to press the input button once for a single instance of the stimulus and twice for a double instance of the stimulus. The model for this task follows this procedure: after noting the stimulus, it waits 100 ms (the inter-stimulus interval used in the original experiment) and then notes whether or not there is a repetition; for the manual response to two instances of the stimulus the model simply presses the button twice. For the vocal response to either stimulus, the model simply assumes a spoken one-syllable utterance to represent the responses "one" and "two." Finally, because the original experiment used a voice recognition system to collect vocal responses, we estimated a parameter to represent the delay in the recognition system, estimated at 300 ms. The final model used the driver model as one thread and the choice model as a second thread. Results were collected from simulations with 8 trials for each condition (auditory-manual, auditory-vocal, visual-manual, visual-vocal) and for each SOA (0, .15, .35, and 1.20 s).

As before, we can examine a detailed process timeline to illustrate how the two tasks are executed together, shown in Figure 12. As mentioned, the driving process involves a repeated firing of four rules that find the near and far points, perform the steering and acceleration (or deceleration) actions, and check for stability on the road. The figure illustrates driving and the choice task performed concurrently, with an assumed SOA of 150 ms. First, for the choice thread, the audition module detects a tone and waits for the stimulus interval to hear the next tone. Once the presence or absence of a second tone is noted, the choice response is made. Meanwhile, the visual stimulus of the lead vehicle's brake light appears after the SOA interval (indicated by the star in the figure); the subsequent visual encoding that occurs as part of the driving thread encodes the presence of this brake light. The driving thread then continues as usual during the choice response motor execution, but when the acceleration adjustments are made for this second driving update (indicated by the #2 arrow), the normal acceleration process is overridden (as per task instructions) and the braking process begins instead. The braking motor movement completes after a delay of 600 ms following motor initiation; meanwhile the choice thread has terminated, and normal driving resumes. Note that after the onset of the visual braking stimulus (indicated by the star), the driving thread twice waits for the procedural resource, causing a 100 ms delay in this thread — and this delay is the source of the PRP effect noted by Levy et al. in their study.

<< Insert Figure 12 approximately here >>

Figure 13(a) shows the human and model reaction times, R^2=.97, *RMSE*=.04, for the choice task and the braking task as a function of SOA, analogous to our earlier analysis of the PRP effect in the dual-choice task. As in the earlier analysis, reaction times for the first task, the choice task, are unaffected by SOA. However, reaction times for the second task, the braking task, are larger for short SOAs: for both human and model data, we see the largest increase for an SOA 0 ms (simultaneous onset), a smaller increase for an SOA of 150 ms, but no increase at SOAs of 350 and 1200 ms. Clearly the cognitive processing required early in the braking task conflicts with that of the choice task for short SOAs, and thus must be delayed for a short time, while this conflict does not occur for larger SOAs.

<< Insert Figure 13 approximately here >>

As another view of these data, Figure 13(b) shows the reaction time for the four combinations of input modalities (auditory and visual) and output modalities (manual and vocal), for both the human drivers in the experiment and the model, R^2=.95, *RMSE*=.04. The visual conditions require more time than the auditory conditions because of contention for the visual modality with the driving task, and the vocal conditions require more time because of the delays of speaking the responses and the subsequent recognition. However, the most interesting aspect of these results is the effect of multitasking. The first two blocks of results show the choice reaction time in the single-task (no braking) and dual-task (while braking) conditions; for both humans and the model, the additional braking task incurs no additional reaction time, essentially mirroring the results for Task 1 in the dual-choice domain. In the third block of results, we also see that input and output modality had no effect on the braking time in the multitasking conditions — in other words, braking performance while multitasking, as indicated in the PRP curves in Figure 13(a), was not affected by input and output modalities.

Driving and Phone Dialing

By combining driving and choice in the previous model, we took one step toward examining a realistic, non-laboratory task domain. In this model, we make both tasks as realistic as possible to address a common, and unfortunately dangerous, real-world scenario: dialing a phone while driving. Numerous studies have examined the effects of phone dialing while driving (e.g., Alm & Nilsson, 1994; Brookhuis, De Vries, & De Waard, 1991; McKnight & McKnight, 1993; Salvucci & Macuga, 2002), and there is general agreement that dialing as a secondary task has significant potential to result in driver distraction with respect to the primary task — that is, decreased performance in the driving task. This domain also benefits our theory in that it provides a first test of threaded cognition for two non-discrete tasks: the continuous driving task, and a dialing task that requires several seconds to complete and thus cannot be completed as a "one-shot" task as could the choice tasks.

We focus our modeling efforts on a recent study of driver distraction (Salvucci, 2001) performed in a realistic driving simulator with a front half of a Nissan 240sx interfaced with a simulated highway environment. In the study, drivers had to steer down an open road while their velocity was held constant, and occasionally dialed a seven-digit phone number in four conditions: *full-manual*, in which they pressed each digit; *speed-manual*, in which they pressed a single "speed number" associated with the phone number; *full-voice*, in which they spoke all seven digits and heard them repeated for confirmation; and *speed-voice*, in which they spoke a single phrase (e.g., "office") associated with the number and heard it repeated for confirmation. Before dialing in all conditions, drivers were required to press a special "power" button to activate the phone, and after the manual conditions drivers were required to press the "send" key to complete dialing. The detailed results appear below, but in short, the *full-manual* condition produced the most distraction, the *speed-manual* condition produced less but significant distraction, and both voice conditions produced no significant distraction.

The previous model of behavior for this task (Salvucci, 2001) required explicit specification of when to switch between the driving and dialing tasks. For present purposes, we began with the driver model used in the previous simulations (driving and choice) — that is, the driver model served as one thread of the full model, just as in the previous simulations. All parameters were kept constant for the current simulations, except that we re-estimated the stability scaling factor (2.5) again to adjust for different simulation conditions. Next, we took the original dialing models for each condition and updated them to the newest version of ACT-R, and at the same time removed any code that specified the explicit switching — allowing threading to handle all switching between tasks. The only parameter assumptions made for the dialing models were that all spoken digits were one syllable and all spoken phrases for the speed-voice condition were three syllables. To begin dialing while driving, the full model simply created the particular dialing control state that corresponded to the desired dialing method, thus creating a new dialing thread that executed along with the driving thread. Results were compiled from three simulation runs, each with 16 dialing trials performed without driving and 16 trials performed while driving.

Figure 14 shows a process timeline for dialing a 7-digit number as two blocks of 3 and 4 digits (following the North American convention of a 3-4 chunking of the phone number). For a new block of numbers, the dialing thread first retrieves a declarative chunk that represents the block, then retrieves the first digit of that block. (This type of declarative representation has been used and independently validated in previous models of list memory (Anderson et al., 1998) and analogy (Salvucci & Anderson, 2001).) Meanwhile, the driving thread performs its processing, ending with a check of vehicle stability. If the vehicle is not sufficiently stable, the driving thread maintains control and iterates until stable; if the vehicle is stable, the dialing thread intercedes. The dialing thread then locates the desired digit to enter and moves the hand to this location while visually encoding the object. In parallel with the button press, the thread retrieves the next digit to enter. When there are no more digits in the block, the dialing thread reiterates block retrieval, at which point the driving thread can re-acquire the visual resource and control the vehicle until stable. Thus, the driving thread controls the vehicle between blocks but not within blocks; a detailed study of the response times for individual digits within the phone number (Salvucci, 2005) suggest that people indeed exhibit this same pattern of behavior.

<< Insert Figure 14 approximately here >>

Figure 15 shows two results from the model simulations and human data: (a) the total time needed to complete the dialing task, both alone and while driving, R^2=.99, *RMSE*=.49; and (b) the vehicle's lateral (side-to-side) velocity, an indicator of vehicle stability and a common measure of driver performance, R^2=.96, *RMSE*=.02. The dialing times in (a) show that the *speed-manual* condition is the fastest and *full-voice* the slowest, both for dialing only and while driving. Perhaps surprisingly, the human drivers required only very little extra time to dial with the added task of driving. The model captures this effect in that, while dialing time slows down slightly because of the driving task, the threaded model is able to interweave the two tasks well enough that the difference is not large. The lateral velocity results in (b) indicate that, as mentioned, the humans showed significant distraction effects for the manual conditions (comparing results for driving only to these conditions) and no significant effect for the voice conditions. The model also captures this effect, primarily because the manual dialing tasks require visual attention and thus must share this resource with the driving task, creating interference and reduced performance.

<< Insert Figure 15 approximately here >>

Driving and the Sentence-Span Task

While most studies of driving and phone dialing focus on the contention of resources (primarily visual) between the tasks, it has also been shown that even primarily cognitive tasks — for example, conversing with another person over the phone — can have detrimental effects on driving performance (see, e.g., Strayer & Johnston, 2001). We now examine this issue of "cognitive distraction" by modeling data reported by Alm and Nilsson (1995) on drivers performing an intensive cognitive "sentence-span" task; this model represents a generalization and threaded version of an earlier model for these data (Salvucci, 2002). By modeling this task, we aim to demonstrate that threaded cognition not only accounts for effects of resource contention for perceptual and motor resources (as in the previous model), but also for effects of contention for cognitive processing — that is, the contention that arises when executing multiple threads.

The driving task in the Alm and Nilsson study was a car-following task where the lead vehicle would sometimes brake suddenly, thus requiring the driver to react and brake in response. The secondary cognitive task, the sentence-span task, was intended as a surrogate for the cognitive load of an intense conversation (given that real-life conversations would be difficult to control for cognitive load across individuals). The sentence-span task (see also Daneman & Carpenter, 1980) involved two stages that included the processing of sentences and the recall of words in these sentences. In the first stage, drivers listened to five sentences of the form “X does Y” — for instance, “The boy brushed his teeth” or “The train bought a newspaper.” After each of these sentences, drivers reported whether the statement was generally sensible. In the second stage, drivers were asked to state the last word of each sentence in the order in which they were presented. For instance, for the sentences “The boy brushed his teeth” and “The train bought a newspaper,” the driver would report “yes” and “no” after each sentence (respectively) and would then report the memorized list “teeth”, “newspaper,” etc. The sentence-span task itself involves two difficult activities, namely judging of sentence sensibility and memorization (and rehearsal) of final words. When combined with driving, the task puts a substantial cognitive load on drivers as they attempt to integrate the tasks.

Our model for this task re-used components from two existing models. First, because the driving task is in essence the same as that in the driving and choice domain (i.e., car-following and braking for a lead vehicle), we imported that model directly for use in the current model, with no changes to rules or parameters. To model the sentence-span task, we based our work on that of Lovett, Daily, and Reder (2000), who modeled a closely related task called the MODS task in which people read strings of letters and digits while memorizing final digits for later recall. Because the task and architectural version differenced from this existing model, we could not import the model directly; however, the original model provided three critical components that were re-used here: (1) the positional representation used to encode memorized items, (2) production rules that perform rehearsal of memorized items, and (3) the basic structure of the production rules that retrieve and report the items in sequence. These rules were then extended to press a button to start the task, to encode a sentence and decide whether it was sensible, and to hear and speak words rather than read and type characters (as in the original MODS model). The process of confirming whether or not the sentence is sensible was not modeled in any detail, but rather the model simply assumes that this process happens during the listening productions and signals a confirmation by firing a confirmation rule. In addition, the model assumes that each sentence component (subject, verb, object) requires one second of speech time with a one-syllable pause (150 ms) in between components. Results were compiled from three simulation runs with four trials each while driving for approximately five minutes. Overall, the sentence-span model creates a heavy cognitive load: during much of the slack time in between encoding auditory stimuli or speaking responses, it rehearses the item list through repeated retrievals of the list, going one by one through all elements of the list and then iterating.

Figure 16 illustrates a segment of a sample process timeline taken while the model executes both the driving and cognitive tasks together. Throughout the timeline, the model interleaves the driving thread with the sentence-span thread. In particular, at the start of the timeline, the sentence-span thread performs a rehearsal retrieval of a list item, and then proceeds to check for the presence of a new audio stimulus (i.e., the spoken sentences streaming in). When a new stimulus is heard (indicated by the star), the sentence-span thread encodes the stimulus and notes it in its declarative structure of the current sentence. In this case, the stimulus represented the end of the sentence, and thus the thread initiates the required spoken response to the sentence (i.e., the sense judgment). As the vocal resource produces this response, the retrieval finally completes — the long delay indicating a new declarative chunk with relatively few rehearsals — and the thread notes the completion in preparation for the next rehearsal retrieval. All the while, the driving thread continues its processing, but the many rule firings required by the sentence-span thread result in longer intervals between driving updates (roughly twice as long as the normal interval). As is evident in the figure, the primary bottleneck in this processing is the procedural resource, because each thread requires frequent procedural rule firings and thus contention for this resource remains high throughout the process.

<< Insert Figure 16 approximately here >>

The Alm and Nilsson study report two significant findings from their analysis of cognitive distraction from the sentence-span task. First, they found a significant effect of the cognitive task on braking performance as measured by the reaction time to the lead-vehicle braking event; specifically, they found that in the presence of the cognitive task, brake reaction time increased by 0.56 s. The threaded cognition model of this combined task produced a very similar increase of 0.54 s. The model did not capture the exact quantitative numbers in the experiment, however: Alm and Nilsson report brake reaction times of approximately 1.6 s and 2.2 s for the driving-only and with-secondary-task conditions (respectively), while the model produced times of 1.00 s and 1.54 s (respectively). Note that our model's prediction of 1.00 s in the driving-only condition comes directly from the driving and choice model, which fit the Levy and Pashler data very well; however, Alm and Nilsson's time of 1.6 s is quite a bit higher for essentially the same task, and we have no explanation for this variance. Nevertheless, the model captures the most important aspect of their data, namely the magnitude of the performance decrement when driving while performing the cognitive task.

Along with this result, Alm and Nilsson predicted that the cognitive task would also significantly affect drivers' ability to maintain a central position on the roadway — that is, they predicted an effect on vehicle position similar to that observed in many studies of driving and phone dialing. However, the empirical data did not show such an effect; Alm and Nilsson do not report specific numbers, only that their statistical analysis yielded no significant effect of the presence of the cognitive task on lateral position. We analyzed the model behavior for effects of lateral position by computing the lateral deviation for both conditions as the root-mean-squared error between the vehicle's position and the lane center (a common measure of driver performance). For this measure, the model produced a lateral deviation of 0.19 m for the driving-only condition and 0.18 m for the with-secondary-task condition, thus finding essentially no effect of secondary task and replicating the findings of Alm and Nilsson's study. The lack of effect on lateral deviation is especially interesting given the presence of the effect for brake reaction time. We attribute this result to the fact that steering updates become, in essence, elongated and slightly less frequent during cognitive distraction, but still occur at regular intervals; thus, the reduced frequency and longer updates translates to additional time for reacting to a braking event, but the regularity of the steering updates (as filtered through vehicle dynamics) does not significantly affect the ability to maintain a central lane position. (This regularity in the cognitive distraction task is much like that in the voice dialing tasks in the driving and dialing domain, both producing no significant lateral effects, in contrast to the long interruptions in the manual dialing tasks that do indeed result in significant lateral effects.)

We should re-emphasize the importance of these simulations with respect to the prediction of contention for cognitive processing resources. Unlike all the modeled tasks we have seen thus far, the combined driving and sentence-span task includes essentially no contention for perceptual and motor resources but includes a secondary task with a high cognitive load. Thus, the "cognitive distraction" that occurs due to the secondary task primarily arises because of contention for cognitive processing. Some cognitive architectures, most saliently EPIC (Meyer & Kieras, 1997), posit a fully parallel cognitive processor with no constraints on processing resources. Such an architecture would thus predict no effects of multitasking performance in this task, because both tasks could execute comfortably in parallel with no perceptual or motor contention to constrain them. In contrast, threaded cognition and the ACT-R cognitive architecture posit a resource-bounded cognitive processor that nicely captures the observed performance decrements that arise in cognitive distraction tasks.

Summary of Model Simulations

The model simulations illustrate the many types of interference that can arise in multitasking behavior, and demonstrate that threaded cognition captures numerous aspects of this behavior. The various domains each emphasize different sources of processing interference. While all the task domains include some degree of cognitive interference due to the reliance on cognition for procedural rule firings, the procedural resource only serves as a primary source of interference in the driving-sentence-span domain, while appearing as a secondary source of interference in the dual-choice and driving-choice domains. The dual-choice domain emphasizes interference in declarative memory retrieval, primarily in the early stages of learning before such retrievals drop out. The tracking-choice and driving-dialing domains emphasize interference effects in perceptual processing, and the tracking-choice domain also demonstrates interference effects in motor processing. In addition, all the domain models demonstrate varying amounts of model re-use and transfer both from earlier work and within the current work; Table 2 summarizes the origins of the models and the changes incorporated for our simulations.

<< Insert Table 2 approximately here >>

General Discussion

Threaded cognition provides a theoretical and computational framework for reasoning about and predicting multitasking behavior. The model simulations emphasize the theory's computational aspects, namely its ability to generate detailed predictions of behavior across a variety of tasks. In this discussion, we expound on the broader theoretical implications of threaded cognition for concurrent multitasking and related types of multitasking behavior.

Theoretical Implications for Concurrent Multitasking

The first implication of our theory of threaded cognition is that concurrent multitasking does not require supervisory or executive processes to manage and schedule multiple task processes. Any approach that requires such processes must specify either how executive processes could be acquired for particular sets of tasks, or how a general executive or supervisory mechanism would operate over all sets of tasks; no previous theory has successfully provided such an explanation. In contrast, threaded cognition allows multitasking to emerge from the interaction of autonomous threads and a threading mechanism for basic resource acquisition and conflict resolution. Our modeling results demonstrate that this straightforward mechanism suffices to account for behavior across a range of multitasking domains, including accounts of when behavior does and does not exhibit multitasking interference.

The second implication of the theory is that both procedural and declarative processes can be sources of dual-task interference, and that declarative processes are an especially prevalent source of interference in the early stages of learning. The procedural resource, as the nexus of processing in our theory, is always present as a potential source of interference because its processing is required to collect results from and initiate processes in other resources. The declarative resource, on the other hand, is most active in early stages of learning because of dependence on memorized task instructions. Over time, however, this dependence reduces and drops out because of the gradual translation of declarative to procedural skill. While the presence of declarative interference has been previously explored (e.g., Carrier and Pashler, 1995), threaded cognition makes explicit the roles of the two resources and how these roles change over time with practice.

The third implication is that the representation of component task skills is simple and parsimonious, free from task-specific knowledge that dictates when and how task switching should occur. With such representations, task skills can be easily integrated or separated, allowing for flexible combinations of single- and dual-task behavior. This aspect of the theory meshes well with the typical methodology for multitasking studies in which component task skills are learned independently: most empirical studies ranging from simple dual-task experiments (e.g., Byrne & Anderson, 2001; Schumacher et al., 1999) to complex studies of secondary tasks while driving (e.g., Alm & Nilsson, 1995; Salvucci, 2001) provide participants with a “warm-up period” in which one or both tasks can be practiced independently of other tasks. Thus, we expect that skill knowledge acquired during this practice time remains uncluttered, without the additional baggage of task-switching knowledge. This being the case, threaded cognition provides the mechanism with which to execute such skills together in a multitasking context.

In fact, a fourth implication of our theory is that, because of the absence of task-specific executive knowledge, practicing two tasks concurrently results in the same performance as practicing the two tasks independently. This implication may seem surprising, but has recently been supported by Ruthruff, Van Selst, Johnston, and Remington (2006) in a study of practice effects in the dual-choice task domain. Thus, threaded cognition accounts for the entire span of learning, from the initial stages of instruction learning to the latest asymptotic stages of highly optimized behavior — an important step toward replacing the oft-derided "homunculus" of control (Altmann, 2003; Logan, 2003; Monsell & Driver, 2000) with a fully specified computational framework.

Implications for Other Aspects of Multitasking

While our theory applies across a range of interesting experimental and real-world domains, threaded cognition focuses on non-deliberative multitasking in which concurrent tasks are performed at the sub-second to second time scale. In contrast, the theory is not meant to capture explicit deliberative multitasking as might arise in some contexts. For example, as discussed earlier, we do not aim to capture the planning-centered, non-greedy types of multitasking that other efforts have addressed (e.g., Freed, 1998; Howes et al., 2004). Nevertheless, the theory does not preclude planning or strategizing about multitasking performance; it simply states that any such higher-level processing must be done by a cognitive thread rather than a specialized parallel executive process, and thus must share resources and perhaps experience interference from other concurrent threads. For example, De Jong (1995) found that in a PRP-like dual-choice task, people respond faster when they can predict which choice task will come first, and they respond slower when their expectation is incorrect. Although threaded cognition itself would not account for this result, such behavior can arise from additional rules that plan for one task over the other (similar to EPIC executive processes forced to run on a serial procedural processor). In essence, the task would be learned as a single thread that performs both tasks in a desired order; in fact, the participants in De Jong's study experienced the dual-task condition immediately with no single-task practice, and were explicitly instructed to pay attention to task ordering. As another example, Logan and Burkell (1986) introduced "stop" and "change" variants of the dual-task paradigm in which a stimulus in one task affects the response to the other task; again, these variants presented two closely tied tasks and thus also biased participants to more of a single-thread representation. In both examples, the resulting representations include rules for higher-level planning and reasoning, but no truly concurrent processing akin to what threaded cognition is intended to address.

Another aspect of such behavior relates to a person's ability to modulate their performance on one task with respect to another in a multitasking context. We often conceptualize multiple tasks as having priorities where one task should receive a larger proportion of processing time than another. However, defining such priorities explicitly can be a difficult endeavor; for instance, in our driving and phone dialing example, driving should clearly have the much higher priority and yet people still dial phones while driving — how can we specify and account for such behavior in our models? Rather than defining priorities per se, our approach allows task representations to incorporate, to some extent, their own requirements for processing time. For example, when our driver model notes that the vehicle is stable, it allows for a short delay before more vehicle control must be performed; by adjusting this delay as external demands change, the driver model can essentially adapt its processing demands over time (see Salvucci, Taatgen, & Kushleyeva, 2006). Such adaptations allow threaded cognition to account for differing processing tradeoffs between two concurrent tasks, such as tradeoffs characterized as points along a performance operating characteristic curve (see, e.g., Navon & Gopher, 1979; Norman & Bobrow, 1975).

Threaded cognition is also not meant to account for multitasking at the minute-to-hour-long time scale, such as multitasking between two long-term work projects but focusing on one for an extended period before switching to the other. For our purposes, this type of multitasking is better characterized as performing sequential tasks with occasional interruptions. These areas of study do not go against the threaded approach presented here; indeed, we believe that these theories and models nicely complement the types of multitasking addressed by threaded cognition, and that they can very likely fit well in the context of a threaded approach.

To illustrate our point, we can consider two bodies of literature in particular through the lens of a threaded approach. First, the sizable literature on task switching (see, e.g., Altmann & Gray, 2002; Rogers & Monsell, 1995; Sohn & Anderson, 2001) has explored various aspects of "switch costs" when performing successive choice tasks — for example, alternating between two distinct choice tasks. For such a domain, the two alternating tasks do not fall under the context of a threaded approach, because threading is intended to capture simultaneous execution of tasks rather than separate, sequential execution. In this case, a model would be required to explicitly change the control state from the current task to the other task, likely requiring a memory retrieval or other costly cognitive function. There have been computational accounts for these switch costs based on theories of memory and recall (e.g., Altmann, 2002; Sohn & Anderson, 2001). Such an account, expressed in the ACT-R framework, can also be expressed in a threaded framework by simply assuming that the model acts as a single thread, explicitly redirecting itself to accomplish alternating tasks represented by different control states.

A second interesting body of literature addresses the issue of interruption. For instance, Trafton et al. (2003) explored people's behavior during the *interruption lag*, or the time interval between an alert of an interruption (e.g., a phone ringing) and the actual interrupting task (e.g., the phone conversation). They found that, when interrupted, people used the interruption lag to prepare for later resumption of the interrupted task (see also Altmann & Trafton, 2002). In particular, they posit that people utilize prospective goal encoding as well as retrospective goal rehearsal to maintain a high activation on the interrupted goal, facilitating later retrieval. Such a model could be neatly expressed in a threaded approach: when the alert occurs, a thread is created that maintains high memory activation by repeated retrievals of the control state (much like our model of the sentence-span task in its rehearsal of the word list). During the interruption lag and even during the interrupting task, such a thread can occasionally perform retrieval and ensure that, upon completion of the interrupting task, the original task can be re-started by retrieval of its control state. Another interpretation might be that the thread of the original task remains during the interrupting task, but the context of the original goal is forgotten, requiring retrieval of this context before resumption. Whatever the approach, the theoretical constructs of Trafton et al.'s theory on interruption and resumption remain wholly intact, but threaded cognition provides a straightforward method to accomplish their posited goal rehearsal — in essence, as a second thread along with the interrupting task thread.

Looking ahead, the ultimate test of threaded cognition arises in the application of the theory across a wide range of task domains. The model simulations presented here represent a first step at demonstrating the theory's breadth and generality using an illustrative representative sampling of domains. As a next step, one might consider the vast space of domains for which cognitive architectural models have been developed, and explore the theory's predictions for when two or more task domains are combined into a single concurrent task. For example, the ACT-R web site[2] contains over 600 publications describing computational models covering over 50 task domains; one might imagine combining any two tasks into a single concurrent task (e.g., lexical processing and spatial navigation, cognitive arithmetic and game playing, etc.), generating predictions of the concurrent task by simply integrating the task models as threads, and then comparing these predictions to (perhaps newly collected) empirical data on the concurrent task. Thus, threaded cognition is an eminently testable theory of multitasking behavior, and we look forward to exploring how well it will generalize to an increasingly diverse array of multitasking domains.

[2] http://act-r.psy.cmu.edu/

References

Alm, H., & Nilsson, L. (1994). Changes in driver behaviour as a function of hands-free mobile phones – A simulator study. *Accident Analysis & Prevention*, *26*, 441-451.

Alm, H., & Nilsson, L. (1995). The effects of a mobile telephone task on driver behaviour in a car following situation. *Accident Analysis & Prevention*, *27*, 707-715.

Altmann, E. M. (2002). Functional decay of memory for tasks. *Psychological Research*, *66*, 287-297.

Altmann, E.M. (2003). Task switching and the pied homunculus: Where are we being led? *Trends in Cognitive Sciences*, *7*, 340–341.

Altmann, E. M., & Gray, W. D. (2002). Forgetting to remember: The functional relationship of decay and interference. *Psychological Science*, *13*, 27-33.

Altmann, E. M., & Trafton, J. G. (2002). Memory for goals: An activation-based model. *Cognitive Science*, *26*, 39-83.

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

Anderson, J. R., Bothell, D., Lebiere, C., & Matessa, M. (1998). An integrated theory of list memory. *Journal of Memory and Language*, *38*, 341-380.

Anderson, J. R., Taatgen, N. A., & Byrne, M. D. (2005). Learning to achieve perfect time sharing: Architectural implications of Hazeltine, Teague, & Ivry (2002). *Journal of Experimental Psychology: Human Perception and Performance, 31*(4), 749-761.

Baddeley, A. D. (1986). *Working memory*. Oxford, England: Oxford University Press.

Baddeley, A., Lewis, V., Eldridge, M., & Thomson, N. (1984). Attention and retrieval from long-term memory. *Journal of Experimental Psychology: General, 113*(4), 518-540.

Broadbent, D. E. (1958). *Perception and communication*. Elmsford, NY: Pergamon Press.

Brookhuis, K. A., De Vries, G., & De Waard, D. (1991). The effects of mobile telephoning on driving performance. *Accident Analysis & Prevention*, *23*, 309-316.

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

Carrier, L. M., & Pashler, H. (1995). Attentional limits in memory retrieval. *Journal of Experimental Psychology: Learning, Memory, and Cogntition, 21*(5), 1339-1348.

Chong, R. S. (1998). Modeling dual-task performance improvement: Casting executive process knowledge acquisition as strategy refinement (Tech. Rep. No. CSE-TR-378-98). Doctoral Dissertation, Department of Computer Science and Engineering, University of Michigan.

Cooper, R., & Shallice, T. (2000). Contention scheduling and the control of routine activities. *Cognitive Neuropsychology*, *17*, 297–338.

Craik, K. J. W. (1948). Theory of the human operator in control systems: II. Man as an element in a control system. *British Journal of Psychology*, *38*, 142-148.

Daneman, M., & Carpenter, P. A. (1980). Individual differences in working memory and reading. *Journal of Verbal Learning and Verbal Behavior*, *19*, 450-466.

De Jong, R. (1995). The role of preparation in overlapping-task performance. *The Quarterly Journal of Experimental Psychology*, *48A*, 2-25.

Freed, M. (1998). Managing multiple tasks in complex, dynamic environments. In *Proceedings of the 1998 National Conference on Artificial Intelligence*, Madison, Wisconsin, 1998.

Gluck, K. A. & Pew, R. W. (Eds.). (2005). *Modeling human behavior with integrated cognitive architectures: Comparison, evaluation, and validation*. Mahwah, NJ: Erlbaum.

Gray, W. D. (Ed.) (2007). *Integrated Models of Cognitive Systems*. Oxford, UK: Oxford University Press.

Graybiel, A. M., & Kimura, M. (1995). Adaptive neural networks in the basal ganglia. In J. C. Houk, J. L. Davis, & D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia* (pp. 103–116). Cambridge, MA: MIT Press.

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

Howes, A., Vera, A., Lewis, R.L., and McCurdy, M. (2004). Cognitive constraint modeling: A formal approach to supporting reasoning about behavior. In Proc. Cognitive Science Society.

Howes, A., Vera, A., Lewis, R.L. (2007). Bounding rational analysis: Constraints on asymptotic performance. In W. D. Gray (Ed.) *Integrated Models of Cognitive Systems* (pp. 403-413). Oxford University Press.

John, B. E. (1996). TYPIST: A theory of performance in skilled typing. *Human-Computer Interaction*, *11*, 321-355.

Johnston, J. C., McCann, R. S., & Remington, R. W. (1995). Chronometric evidence for two types of attention. *Psychological Science, 6*(6), 365-369.

Jones, R. M., Laird, J. E., Nielsen P. E., Coulter, K., Kenny, P., & Koss, F. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine*, *20*, 27-42.

Just, M. A., Carpenter, P. A., & Varma, S. (1999). Computational modeling of high-level cognition and brain function. *Human Brain Mapping*, *8*, 128-136.

Keele, S. W. (1973). *Attention and human performance*. Pacific Palisades, CA: Goodyear.

Kieras, D. E. (2007). Control of cognition. In W.D. Gray (Ed.) *Integrated Models of Cognitive Systems* (pp. 327-355). Oxford University Press.

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

Kucera, H., & Francis, W. N. (1967). *Computational analysis of present-day American English*. Providence, RI: Brown University Press.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1-64.

Lallement, Y., & John, B. E. (1998). Cognitive architecture and modeling idiom: An examination of three models of the Wickens task. In M. A. Gernsbacher & S. J. Derry (Eds.), *Proceedings of the twentieth annual conference of the Cognitive Science Society* (pp. 597-602). Hillsdale, NJ: Erlbaum.

Lee, J. D., McGehee D. V., Brown T. L., Reyes, M. L. (2002). Collision warning timing, driver distraction, and driver response to imminent rear-end collisions in a high fidelity driving simulator. *Human Factors*, *44*, 314-334.

Levy, J., Pashler, H., & Boer, E. (2006). Central interference in driving: Is there any stopping the psychological refractory period? *Psychological Science*, *17*, 228-235.

Lewis, R. L., & Vasishth, S. (2005). An activation-based model of sentence processing as as skilled memory retrieval. *Cognitive Science, 29*, 375-419.

Liu, Y. (1996). Queueing network modeling of elementary mental processes. *Psychological Review*, *103*, 116-136.

Liu, Y., Feyen, R., & Tsimhoni, O. (2005). Queueing Network-Model Human Processor (QN-MHP): A computational architecture for multi-task performance in human-machine systems. *ACM Transactions on Computer-Human Interaction*, *13*, 37-70.

Logan, G. D. (2003). Executive control of thought and action: In search of the wild homunculus. *Current Directions in Psychological Science*, *12*, 45–48.

Logan, G. D., & Burkell, J. (1986). Dependence and independence in responding to double stimulation: Comparison of stop, change, and dual-task paradigms. *Journal of Experimental Psychology: Human Perception and Performance*, *12*, 549-563.

Lovett, M. C., Daily, L. Z., & Reder, L. M. (2000). A source activation theory of working memory: Cross-task prediction of performance in ACT-R. *Journal of Cognitive Systems Research*, *1*, 99-118.

Martin-Emerson, R., & Wickens, C. D. (1992). The vertical visual field and implications for the head-up display. In *Proceedings of the Thirty-Sixth Annual Symposium of the Human Factors Society* (pp. 1408-1412). Santa Monica, CA: Human Factors Society.

McKnight, A. J., & McKnight, A. S. (1993). The effect of cellular phone use upon driver attention. *Accident Analysis & Prevention*, *25*, 259-265.

Meyer, D. E., & Kieras, D. E. (1997). A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review*, *104*, 3-65.

Meyer, D. E., & Kieras, D. E. (1997-b). A computational theory of executive cognitive processes and multiple-task performance: Part 2. Accounts of psychological refractory period phenomena. *Psychological Review*, *104*, 749-791.

Monsell, S., & Driver, J. (2000). Banishing the control homunculus. In S. Monsell & J. Driver (Eds.), *Control of Cognitive Processes: Attention and Performance XVIII* (pp. 3-32). Cambridge, MA: MIT Press.

Navon, D., & Gopher, D. (1979). On the economy of the human-processing system. *Psychological Review*, *86*, 214-255.

Newell, A. & Simon, H. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Norman, D. A., & Bobrow, D. G. (1975). On data-limited and resource-limited processes. *Cognitive Psychology*, *7*, 44-64.

Norman, D. A., & Shallice, T. (1986). Attention to action: Willed and automatic control of behavior. In R. J. Davidson, G. E. Schwartz, & D. Shapiro (Eds.), *Consciousness and Self-Regulation* (pp. 1-18). New York: Plenum.

O’Reilly, R. C., & Frank, M. J. (2006). Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation*, *18*, 283–328.

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

Pashler, H., & Johnston, J. C. (1998). Attentional limitations in dual-task performance. In H. Pashler (Ed.), *Attention* (pp. 155-189). Hove, UK: Taylor & Francis.

Reichle, E. D., Pollatsek, A., Fisher, D. L., & Rayner, K. (1998). Toward a model of eye movement control in reading. *Psychological Review*, *105*, 125-157.

Rogers, R. D., & Monsell, S. (1995). Costs of a predictable switch between simple cognitive tasks. *Journal of Experimental Psychology: General*, *124*, 207-231.

Rohrer, D., & Pashler, H. E. (2003). Concurrent task effects on memory retrieval. *Psychonomic Bulletin & Review, 10*(1), 96-103.

Rumelhart, D.E. (1980). Schemata: The building blocks of cognition. In R.J. Spiro, B.Bruce, & W.F. Brewer (eds.), *Theoretical Issues in Reading and Comprehension*. Hillsdale, NJ: Erlbaum.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., & Hinton, G. E. (1986). Schemata and sequential thought processes in PDP models. In D.E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing, volume 2* (pp. 7-57). Cambridge: MIT Press.

Ruthruff, E., Van Selst, M., Johnston, J. C., & Remington, R. (2006). How does practice reduce dual-task interference: Integration, automatization, or just stage-shortening? *Psychological* Research, *70*, 125-142.

Salvucci, D. D. (2001). Predicting the effects of in-car interface use on driver performance: An integrated model approach. *International Journal of Human-Computer Studies*, *55*, 85-107.

Salvucci, D. D. (2001-b). An integrated model of eye movements and visual encoding. *Cognitive Systems Research*, *1*, 201-220.

Salvucci, D. D. (2002). Modeling driver distraction from cognitive tasks. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society* (pp. 792-797). Mahwah, NJ: Lawrence Erlbaum Associates.

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

Salvucci, D. D. (2006). Modeling driver behavior in a cognitive architecture. *Human Factors*, *48*, 362-380.

Salvucci, D. D., & Anderson, J. R. (2001). Integrating analogical mapping and general problem solving: The path-mapping theory. *Cognitive Science*, *25*, 67-110.

Salvucci, D. D., & Gray, R. (2004). A two-point visual control model of steering. *Perception*, *33*, 1233-1248.

Salvucci, D. D., & Macuga, K. L. (2002). Predicting the effects of cellular-phone dialing on driver performance. *Cognitive Systems Research*, *3*, 95-102.

Salvucci, D. D., Taatgen, N. A., & Kushleyeva, Y. (2006). Learning when to switch tasks in a dynamic multitasking environment. In *Proceedings of the Seventh International Conference on Cognitive Modeling* (pp. 268-273). Trieste, Italy: Edizioni Goliardiche.

Schumacher, E. H., Lauber, E. J., Glass, J. M., Zurbriggen, E. L., Gmeindl, L., Kieras, D. E., & Meyer, D. E. (1999). Concurrent response-selection processes in dual-task performance: Evidence for adaptive executive control of task scheduling. *Journal of Experimental Psychology: Human Perception and Performance*, *25*, 791-814.

Schumacher, E. H., Seymour, T. L., Glass, J. M., Fencsik, D. E., Lauber, E. J., Kieras, D. E., et al. (2001). Virtually perfect time sharing in dual-task performance: uncorking the central cognitive bottleneck. *Psychological Science, 12*(2), 101-108.

Sohn, M.-H., & Anderson, J. R. (2001). Task preparation and task repetition: Two-component model of task switching. *Journal of Experimental Psychology: General*, *130*, 764-778.

Spelke, E., Hirst, W., & Neisser, U. (1976). Skills of divided attention. *Cognition, 4*, 215-230.

Strayer, D. L., & Johnston, W. A. (2001). Driven to distraction: Dual-task studies of simulated driving and conversing on a cellular telephone. *Psychological Science*, *12*, 462-466.

Taatgen, N.A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421-455.

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

Taatgen, N. A., van Rijn, H., & Anderson, J. R. (2007). An integrated theory of prospective time interval estimation: The role of cognition, attention and learning. *Psychological Review*, *114*, 577-598.

Telford, C. W. (1931). The refractory phase of voluntary and associative response. *Journal of Experimental Psychology*, *14*, 1-35.

Trafton, J. G., Altmann, E. M., Brock, D. P., & Mintz, F. E. (2003). Preparing to resume an interrupted task: Effects of prospective goal encoding and retrospective rehearsal. *International Journal of Human-Computer Studies*, *58*, 583-603.

Welford, A. T. (1952). The "psychological refractory period" and the timing of high speed performance: A review and a theory. *British Journal of Psychology*, *43*, 2-19.

Wickens, C. D. (1984). Processing resources in attention. In R. Parasuraman, J. Beatty, & R. Davies (Eds.), *Varieties of Attention* (pp. 63-101). New York: Wiley.

Wickens, C. D. (2002). Multiple resources and performance prediction. *Theoretical Issues in Ergonomics Science*, *3*, 159-177.

Acknowledgments

This work was supported by ONR Grant #N00014-03-1-0036 to the first author and ONR Grant #N00014-06-1-0055 to the second author. We thank Erik Altmann, Erik Reichle, Stefani Nellen, Yelena Kushleyeva, John Anderson, and three anonymous reviewers for their many comments and suggestions; Ron Chong for sharing his implementation of the tracking task; and Susan Chipman for her generous support.

Tables

Table 1.: Overview of theoretical coverage provided by the chosen modeling domains.

	Dual Choice	Tracking & Choice	Reading & Dictation	Driving & Secondary Tasks
Dual-Task Interference	X	X		X
Lack of Dual-Task Interference	X		X	X
Practice Effects	X		X	
Laboratory Tasks	X	X	X	
Real-World Tasks				X
Model Transfer	X			X

Table 2.: Summary of model origins and parameter settings.

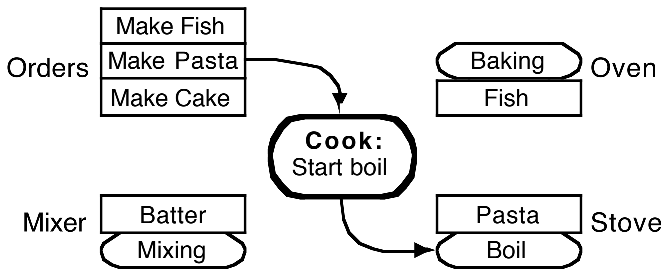
Dual Choice	Choice model based on Taatgen (2005) • No parameters estimated; all values taken from Taatgen (2005) and Byrne & Anderson (2001)
Tracking & Choice	Tracking model newly developed • No parameters estimated; all parameters left at default values Choice model adapted from Dual Choice study • Model changed to identify arrow stimuli instead of previous stimuli • No parameters estimated
Reading & Dictation	Reading model based on Lewis & Vasishth (2005) • Model implementation a simplified version of original (which would require too much time to run for simulated days) • Set one parameter, latency factor, so that the timing of all model actions is the same as in the original model Dictation model newly developed • Estimated 2 parameters: motor action and procedural learning speed
Driving & Choice	Driving model based on Salvucci (2005) • Model implementation changed to new version of ACT-R • Model changed to break out of normal delay due to braking task • Estimated 3 parameters: steering scaling factor, stability scaling factor, and accelerator-to-brake time Choice model adapted from Dual Choice study • Model changed to accept dual-instance stimulus • Estimated 1 parameter: voice recognition delay
Driving & Dialing	Driving model taken from Driving & Choice study • Model changed to remove braking task (not applicable here) • Estimated 1 parameter: stability scaling factor Dialing model taken from Salvucci (2001) • Assumed that digit is one syllable and phrases are three syllables • No new parameters estimated
Driving & Sentence-Span	Driving model taken from Driving & Choice study • No new parameters estimated Sentence-span model derived from Lovett et al. (2000) • Model uses representation and some rules from earlier model

Figure Captions

Figures

Figure 1.

(a)


Orders
Make Fish
Make Pasta
Make Cake
Baking
Fish
Oven
Cook:
Start boil
Mixer
Batter
Mixing
Pasta
Boil
Stove

(b)

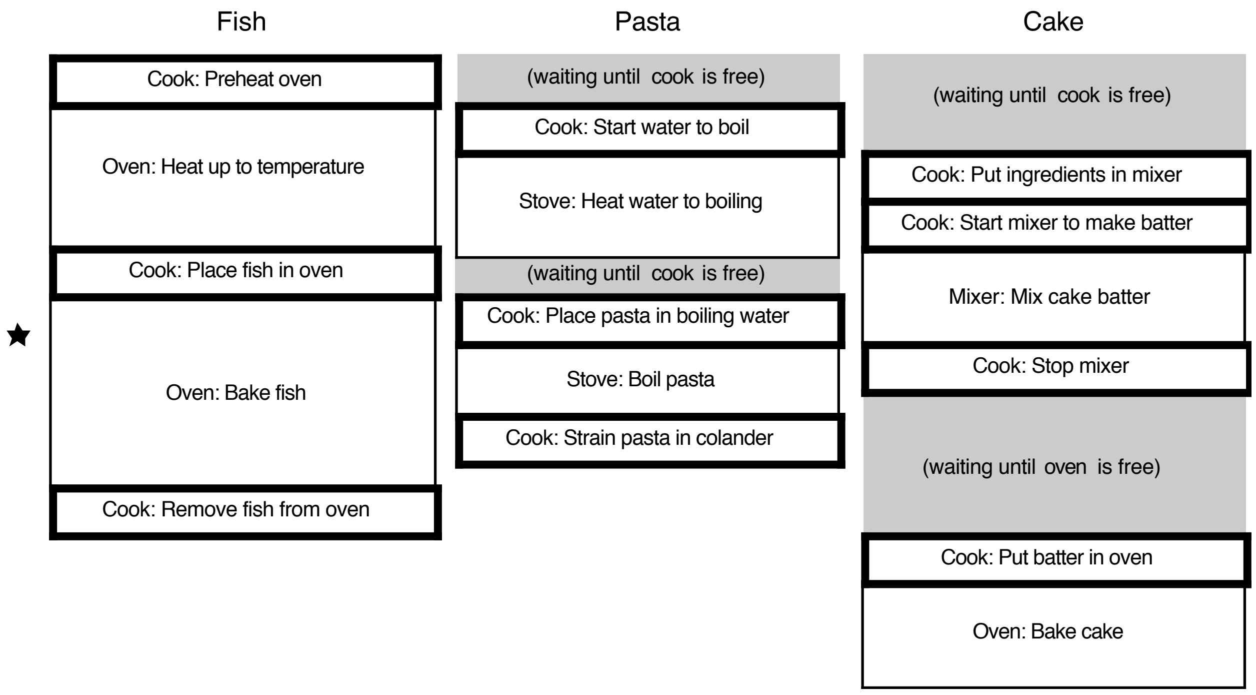
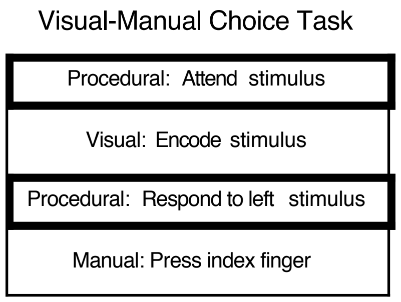
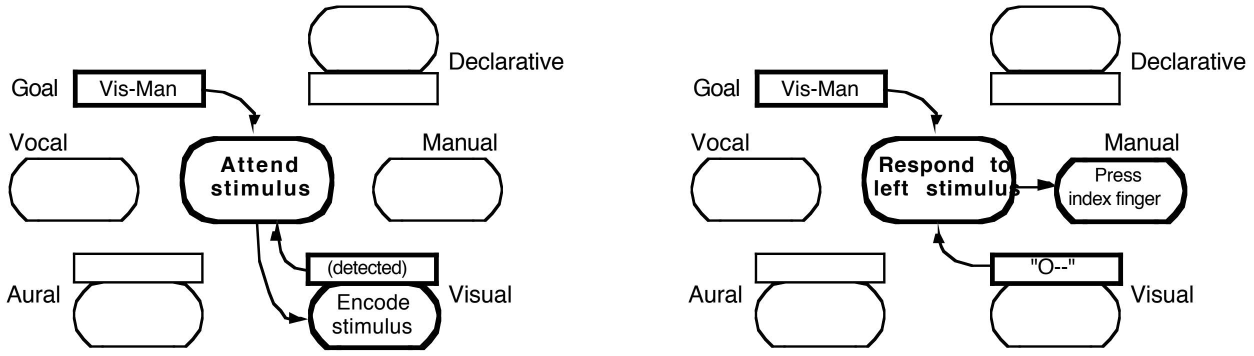

Fish
Pasta
Cake
Cook: Preheat oven
Oven: Heat up to temperature
Cook: Place fish in oven
Oven: Bake fish
Cook: Remove fish from oven
(waiting until cook is free)
Cook: Start water to boil
Stove: Heat water to boiling
(waiting until cook is free)
Cook: Place pasta in boiling water
Stove: Boil pasta
Cook: Strain pasta in colander
(waiting until cook is free)
Cook: Put ingredients in mixer
Cook: Start mixer to make batter
Mixer: Mix cake batter
Cook: Stop mixer
(waiting until oven is free)
Cook: Put batter in oven
Oven: Bake cake

Figure 2.

(a)


Visual-Manual Choice Task
Procedural: Attend stimulus
Visual: Encode stimulus
Procedural: Respond to left stimulus
Manual: Press index finger

(b)

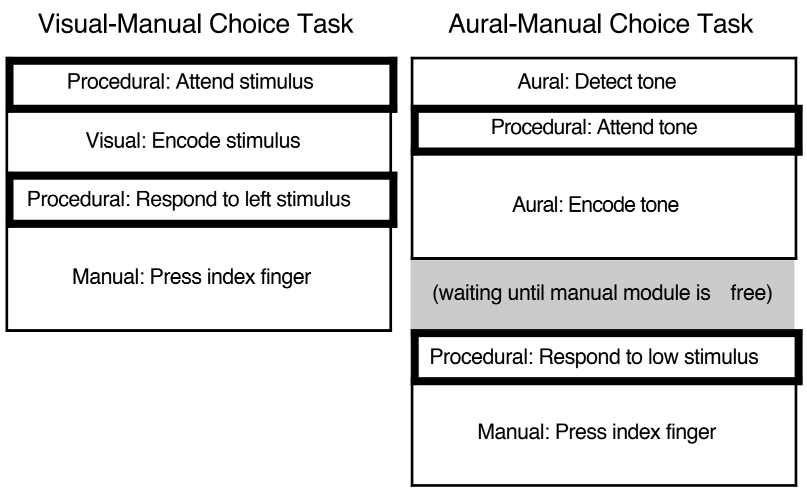


t = 0-50 ms
The presence of a stimulus detected by the visual resource combined with the current goal activate the first rule, which issues a request to the visual resource to encode the full visual stimulus.

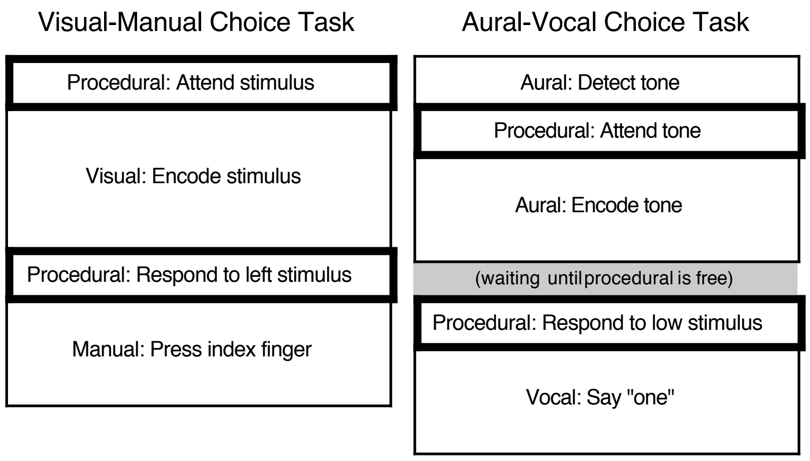
t = 135-185 ms
When visual encoding is complete, the encoded stimulus and the goal activate the second rule, which responds to the leftward position of the stimulus by pressing the index finger.

Figure 3.

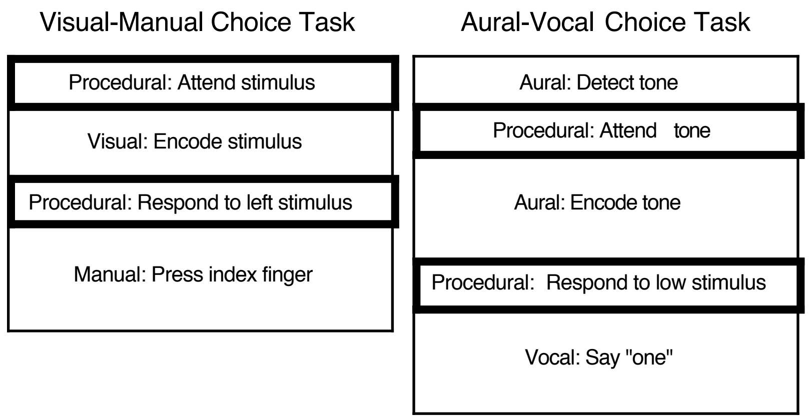
(a)


Visual-Manual Choice Task
Procedural: Attend stimulus
Visual: Encode stimulus
Procedural: Respond to left stimulus
Manual: Press index finger
Aural-Manual Choice Task
Aural: Detect tone
Procedural: Attend tone
Aural: Encode tone
(waiting until manual module is free)
Procedural: Respond to low stimulus
Manual: Press index finger

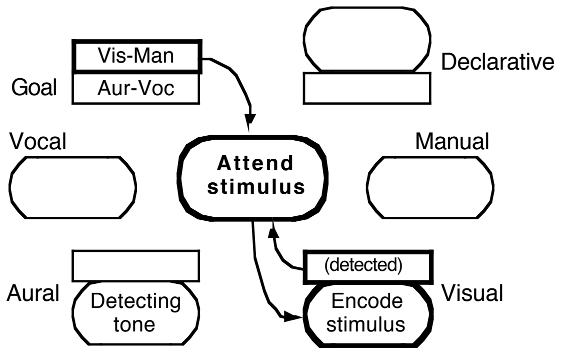
(b)


Visual-Manual Choice Task
Procedural: Attend stimulus
Visual: Encode stimulus
Procedural: Respond to left stimulus
Manual: Press index finger
Aural-Vocal Choice Task
Aural: Detect tone
Procedural: Attend tone
Aural: Encode tone
(waiting until procedural is free)
Procedural: Respond to low stimulus
Vocal: Say "one"

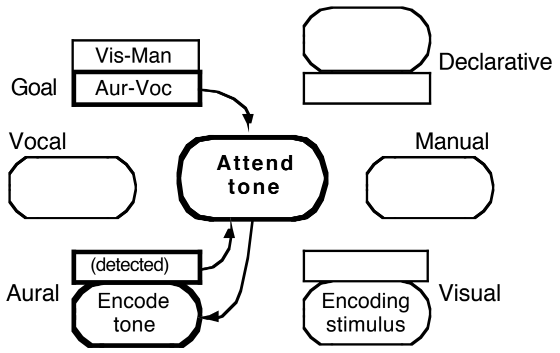
(c)


Visual-Manual Choice Task
Procedural: Attend stimulus
Visual: Encode stimulus
Procedural: Respond to left stimulus
Manual: Press index finger
Aural-Vocal Choice Task
Aural: Detect tone
Procedural: Attend tone
Aural: Encode tone
Procedural: Respond to low stimulus
Vocal: Say "one"

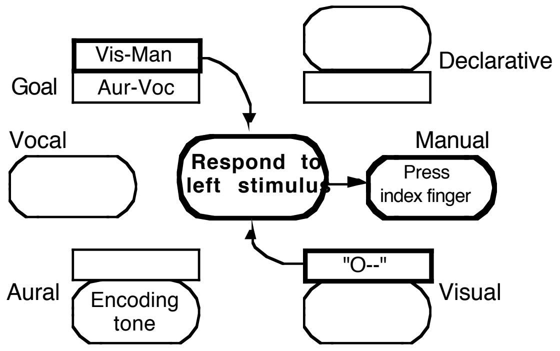
(d)



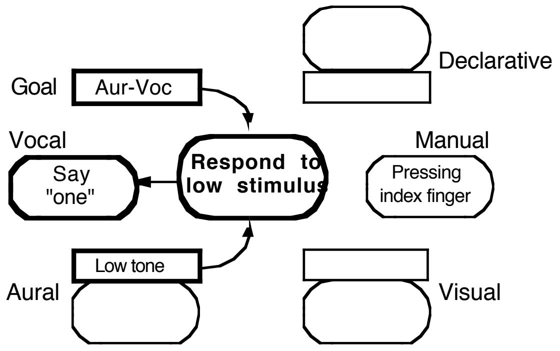
t = 0-50 ms
The presence of a stimulus detected by the visual resource combined with the Visual-Manual goal trigger the rule that initiates visual encoding. Meanwhile, the aural module is detecting the aural tone.



t = 50-100 ms
The detected tone and the Aural-Vocal goal activate the rule that initiates encoding of the tone (i.e., determines the tone's pitch as needed for the choice response).



t = 135-185 ms
When visual encoding is complete, the pattern placed in the visual buffer and the Visual-Manual goal activate the rule that initiates the finger press. The Visual-Manual goal has completed and terminates.



t = 220-270 ms
When aural encoding is complete, the low tone and the Aural-Vocal goal activate the rule that initiates the vocal response "one" associated with the tone. The Aural-Vocal goal then terminates.

Figure 4.

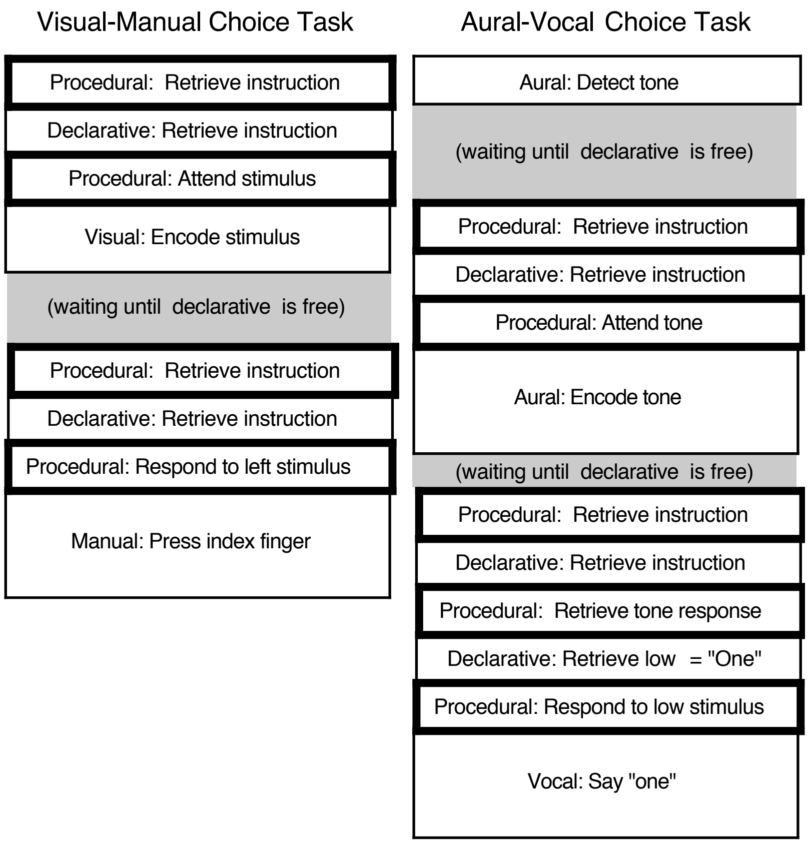

Visual-Manual Choice Task
Procedural: Retrieve instruction
Declarative: Retrieve instruction
Procedural: Attend stimulus
Visual: Encode stimulus
(waiting until declarative is free)
Procedural: Retrieve instruction
Declarative: Retrieve instruction
Procedural: Respond to left stimulus
Manual: Press index finger
Aural-Vocal Choice Task
Aural: Detect tone
(waiting until declarative is free)
Procedural: Retrieve instruction
Declarative: Retrieve instruction
Procedural: Attend tone
Aural: Encode tone
(waiting until declarative is free)
Procedural: Retrieve instruction
Declarative: Retrieve instruction
Procedural: Retrieve tone response
Declarative: Retrieve low = "One"
Procedural: Respond to low stimulus
Vocal: Say "one"

Figure 5.

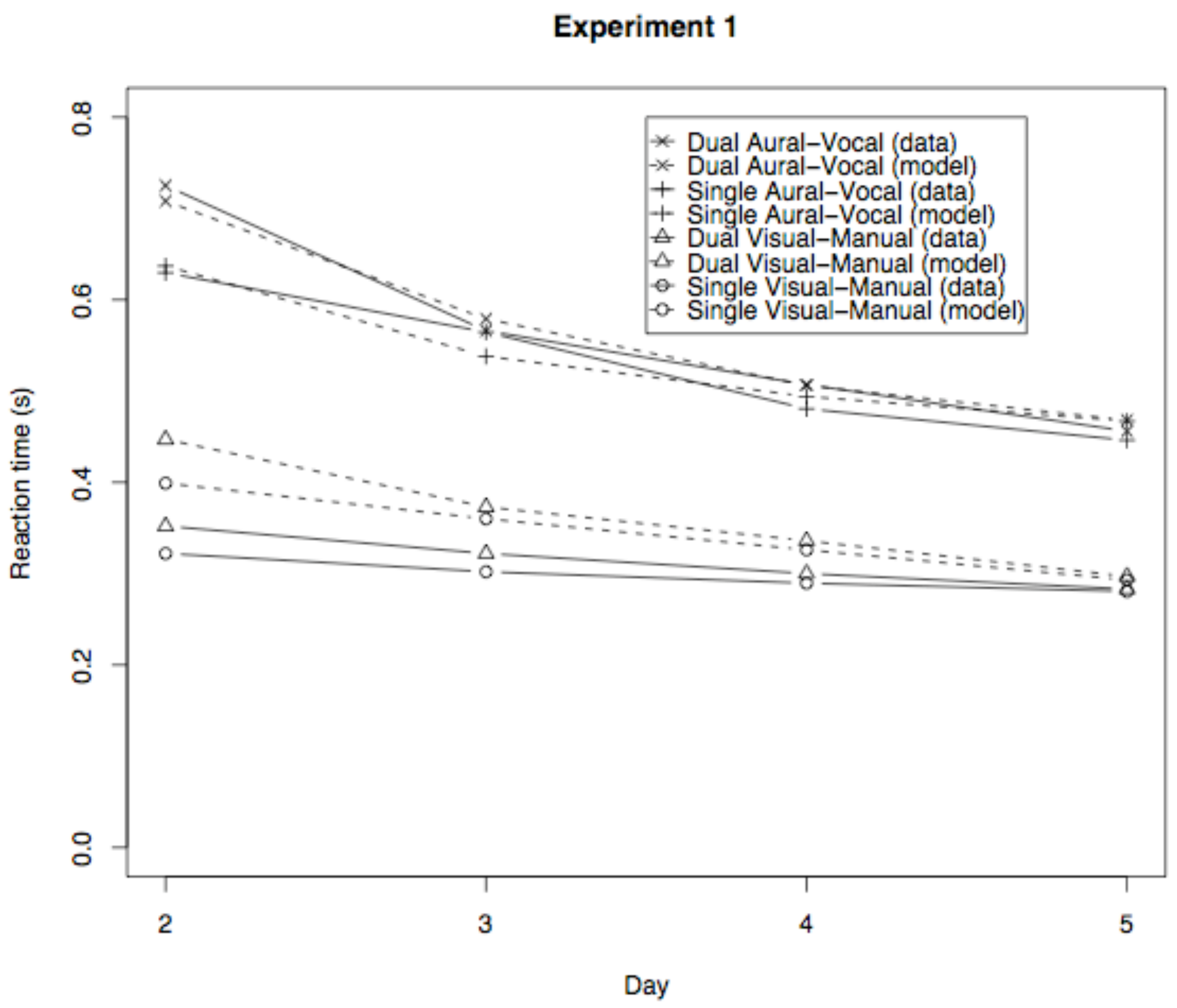

Experiment 1
Dual Aural–Vocal (data)
Dual Aural–Vocal (model)
Single Aural–Vocal (data)
Single Aural–Vocal (model)
Dual Visual–Manual (data)
Dual Visual–Manual (model)
Single Visual–Manual (data)
Single Visual–Manual (model)
Reaction time (s)
0.0
0.2
0.4
0.6
0.8
Day
2
3
4
5

Figure 6.

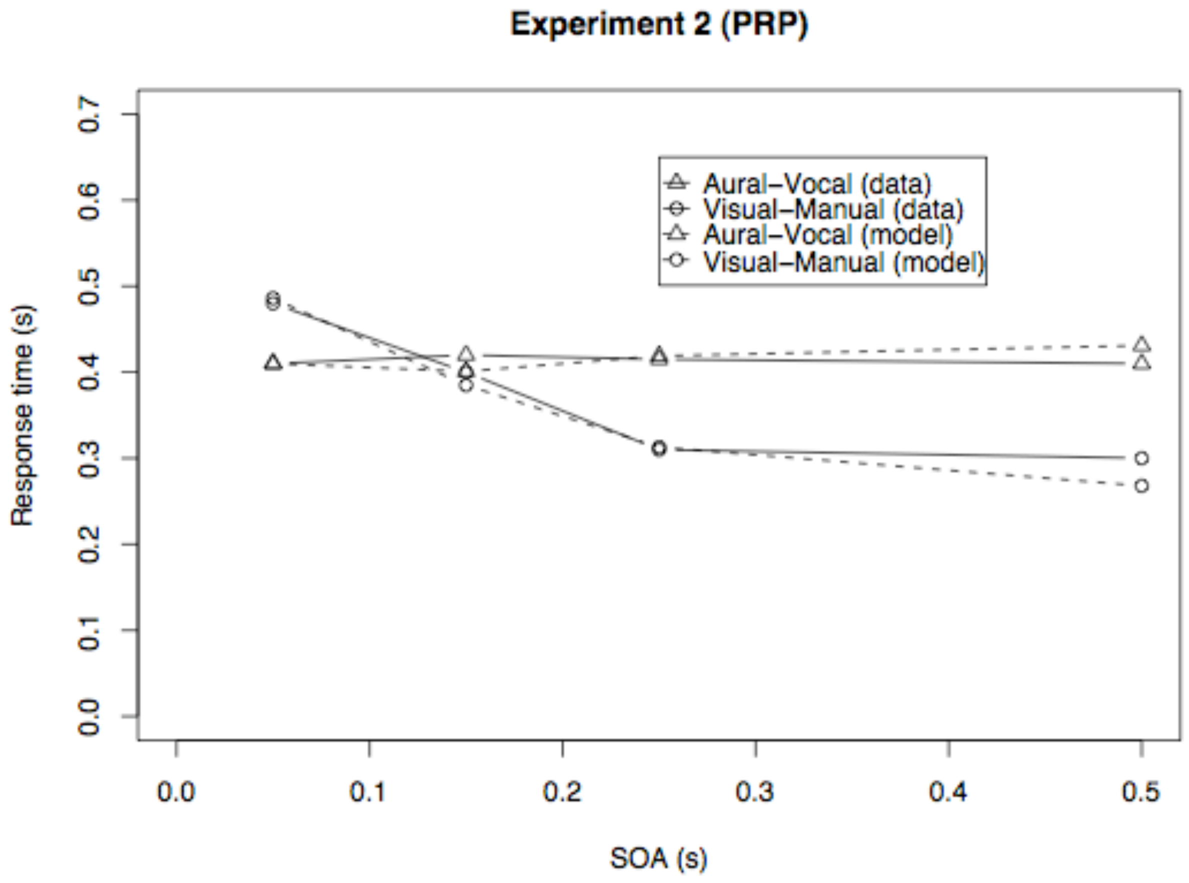

Experiment 2 (PRP)
Aural–Vocal (data)
Visual–Manual (data)
Aural–Vocal (model)
Visual–Manual (model)
Response time (s)
SOA (s)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

Figure 7.

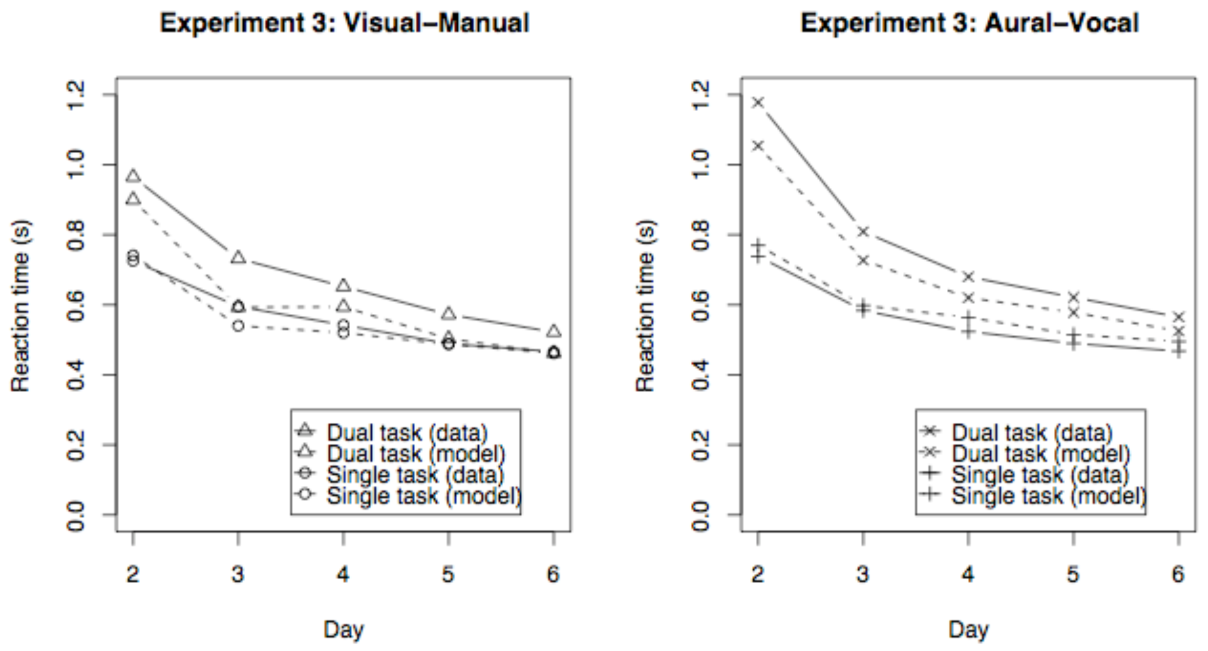

Experiment 3: Visual–Manual
Reaction time (s)
Day
Dual task (data)
Dual task (model)
Single task (data)
Single task (model)
Experiment 3: Aural–Vocal
Reaction time (s)
Day
Dual task (data)
Dual task (model)
Single task (data)
Single task (model)

Figure 8.

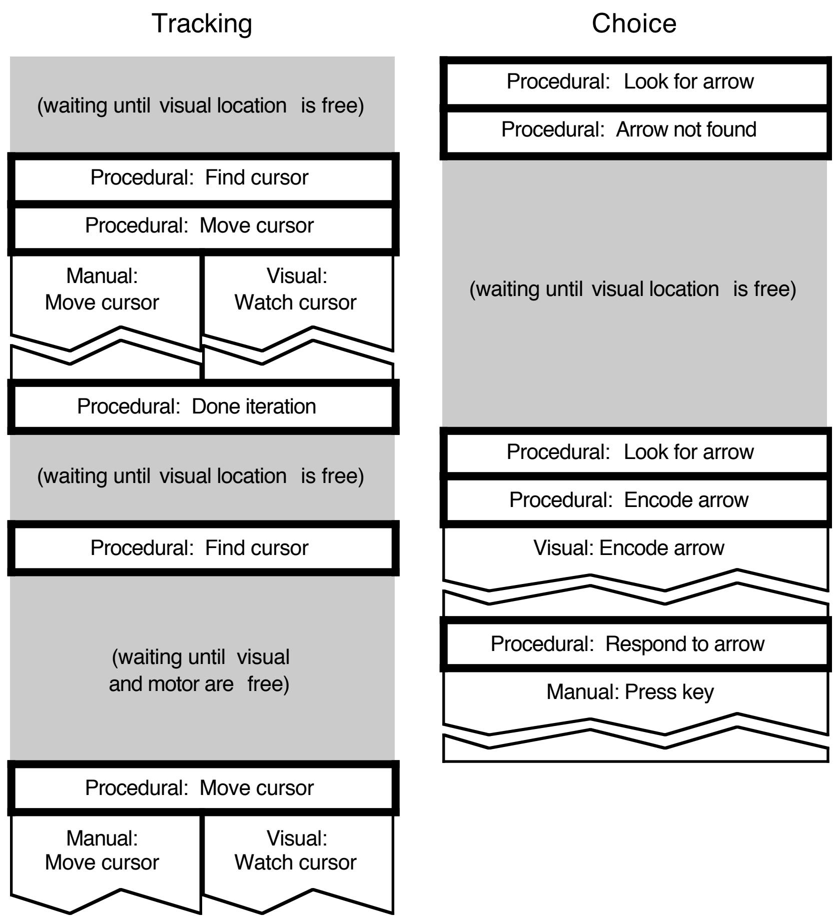
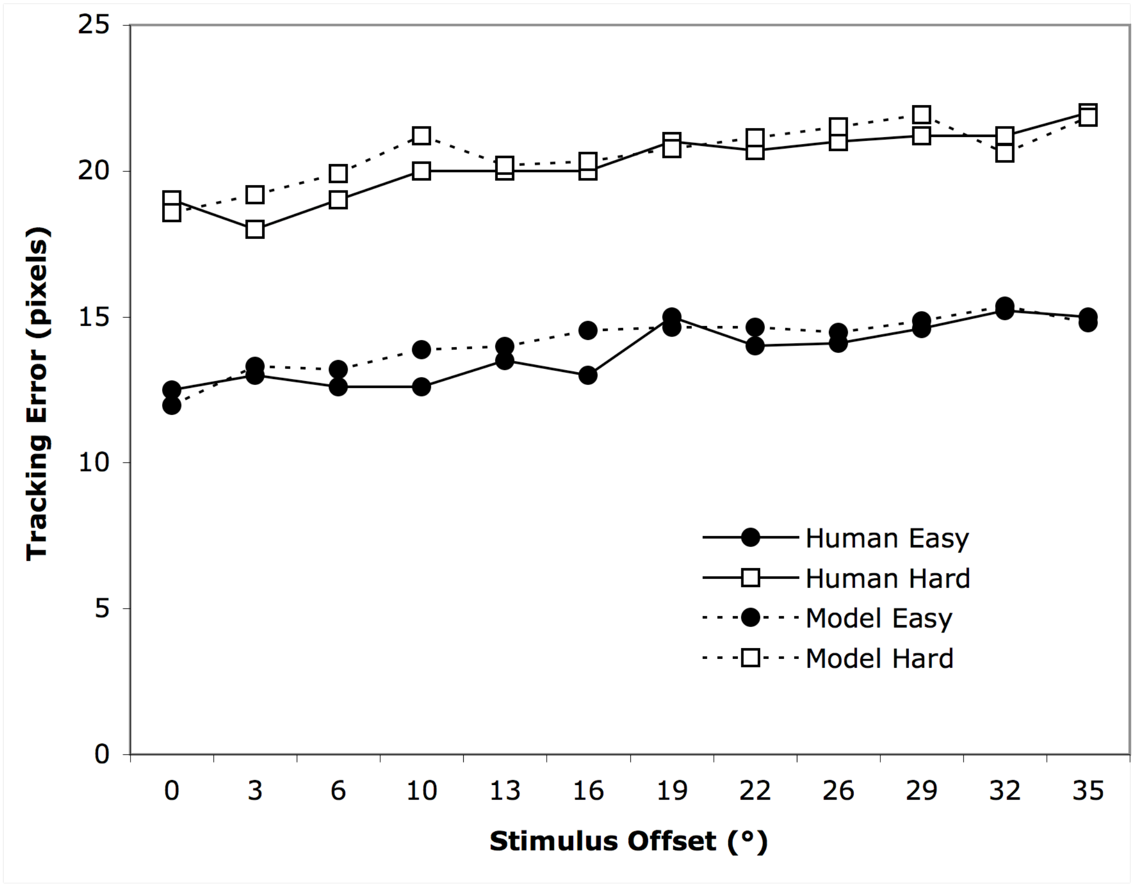

Tracking
Choice
(waiting until visual location is free)
Procedural: Find cursor
Procedural: Move cursor
Manual: Move cursor
Visual: Watch cursor
Procedural: Done iteration
(waiting until visual location is free)
Procedural: Find cursor
(waiting until visual and motor are free)
Procedural: Move cursor
Manual: Move cursor
Visual: Watch cursor
Procedural: Look for arrow
Procedural: Arrow not found
(waiting until visual location is free)
Procedural: Look for arrow
Procedural: Encode arrow
Visual: Encode arrow
Procedural: Respond to arrow
Manual: Press key

Figure 9.

(a) error



(b) time

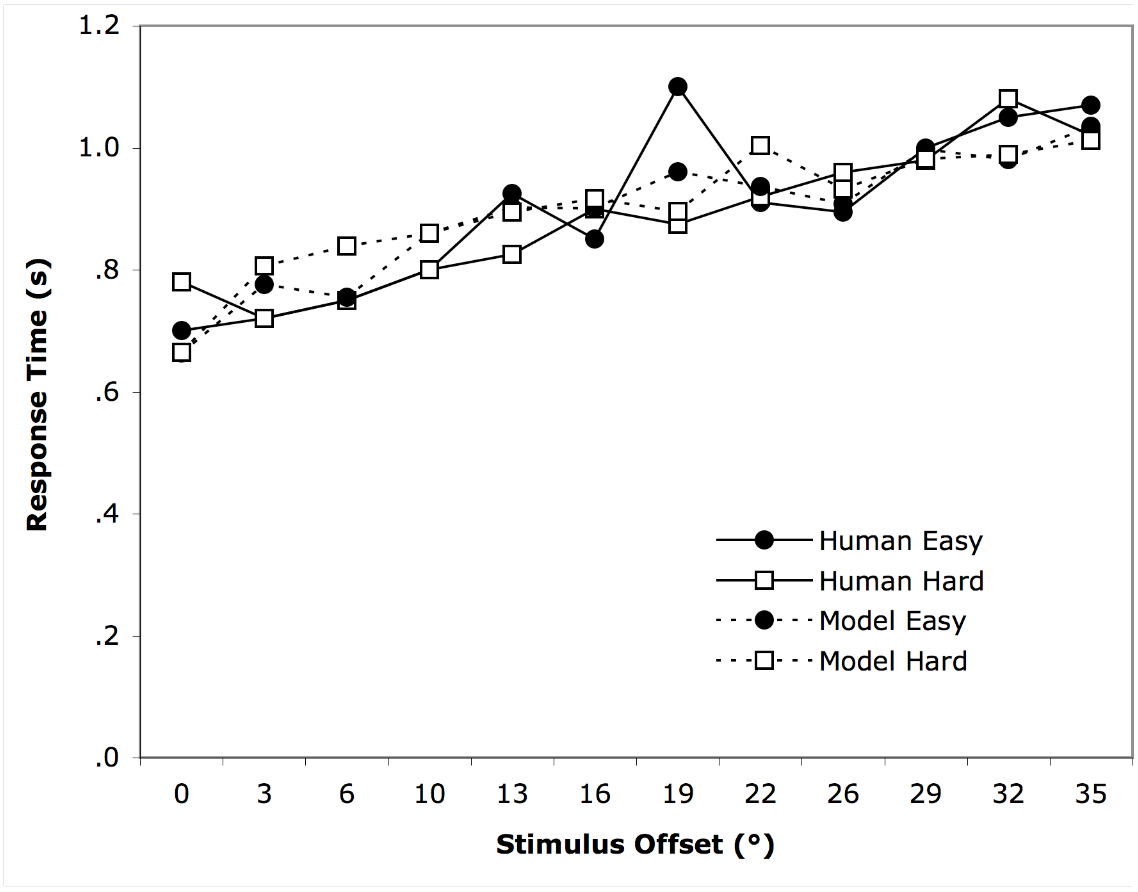

1.2
1.0
.8
.6
.4
.2
.0
Response Time (s)
0
3
6
10
13
16
19
22
26
29
32
35
Stimulus Offset (°)
Human Easy
Human Hard
Model Easy
Model Hard

Figure 10.

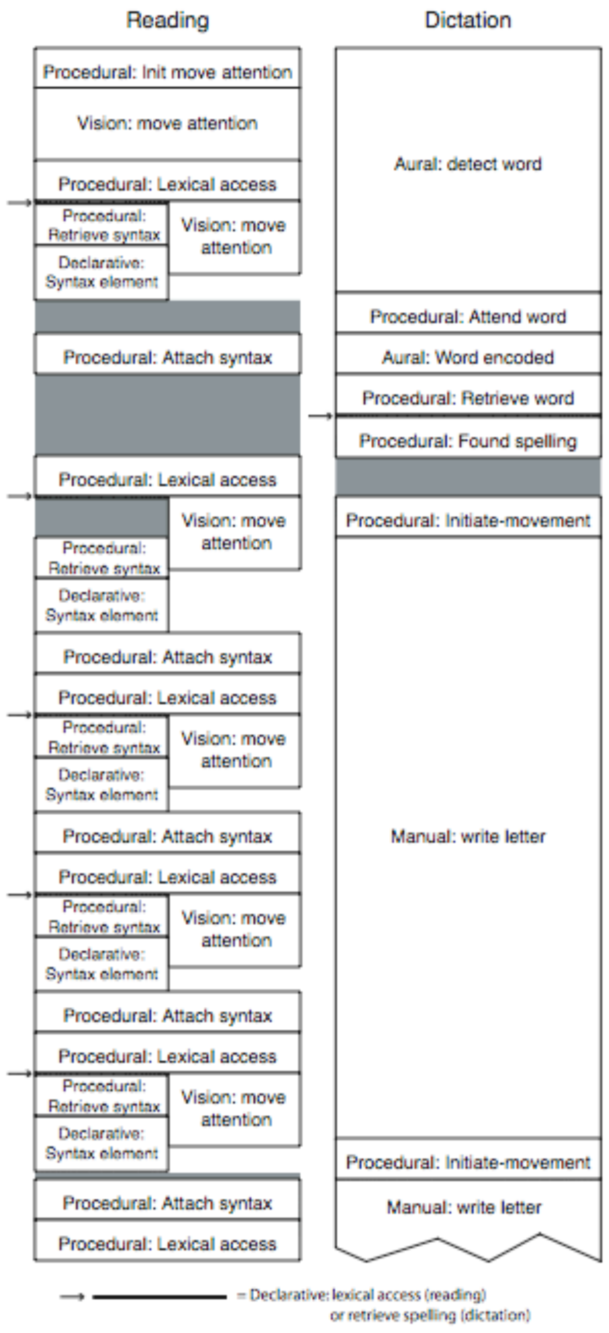

Reading
Dictation
Procedural: Init move attention
Vision: move attention
Procedural: Lexical access
Procedural: Retrieve syntax
Declarative: Syntax element
Vision: move attention
Procedural: Attach syntax
Aural: detect word
Procedural: Attend word
Aural: Word encoded
Procedural: Retrieve word
Procedural: Found spelling
Procedural: Initiate-movement
Manual: write letter
Procedural: Initiate-movement
Manual: write letter
= Declarative: lexical access (reading) or retrieve spelling (dictation)

Figure 11.

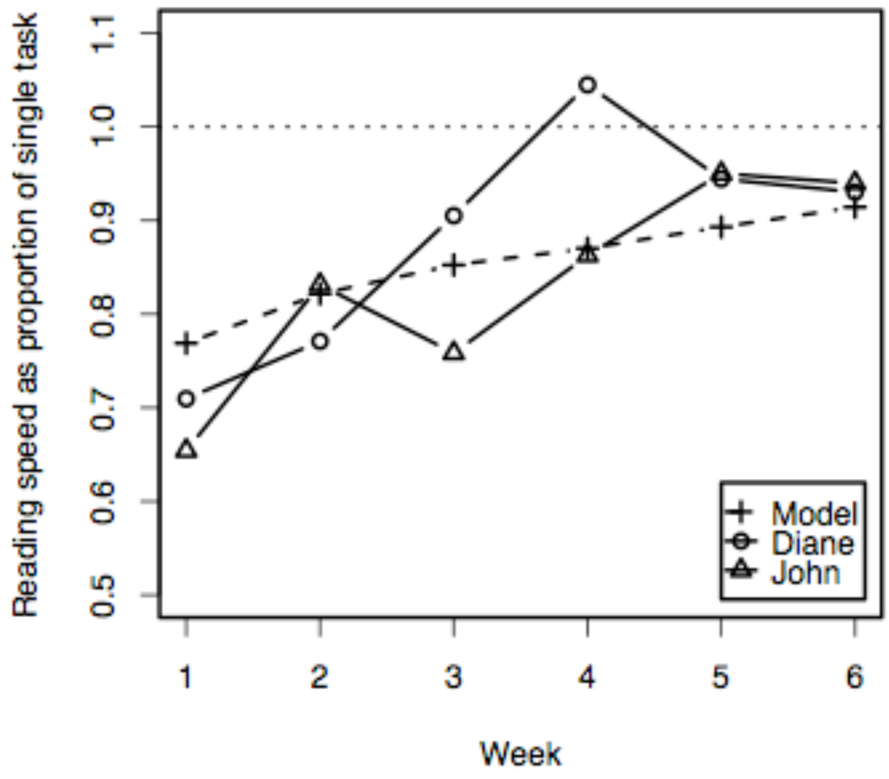

Reading speed as proportion of single task
0.5
0.6
0.7
0.8
0.9
1.0
1.1
Week
1
2
3
4
5
6
+ Model
o Diane
△ John

Figure 12.

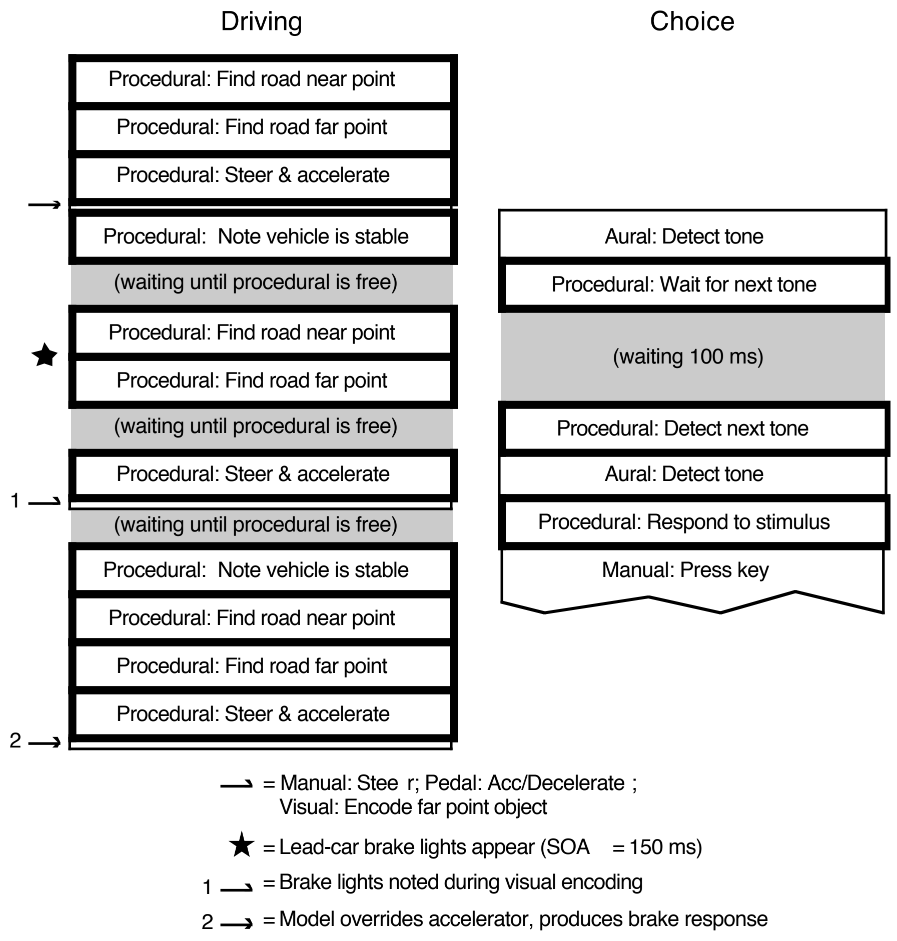
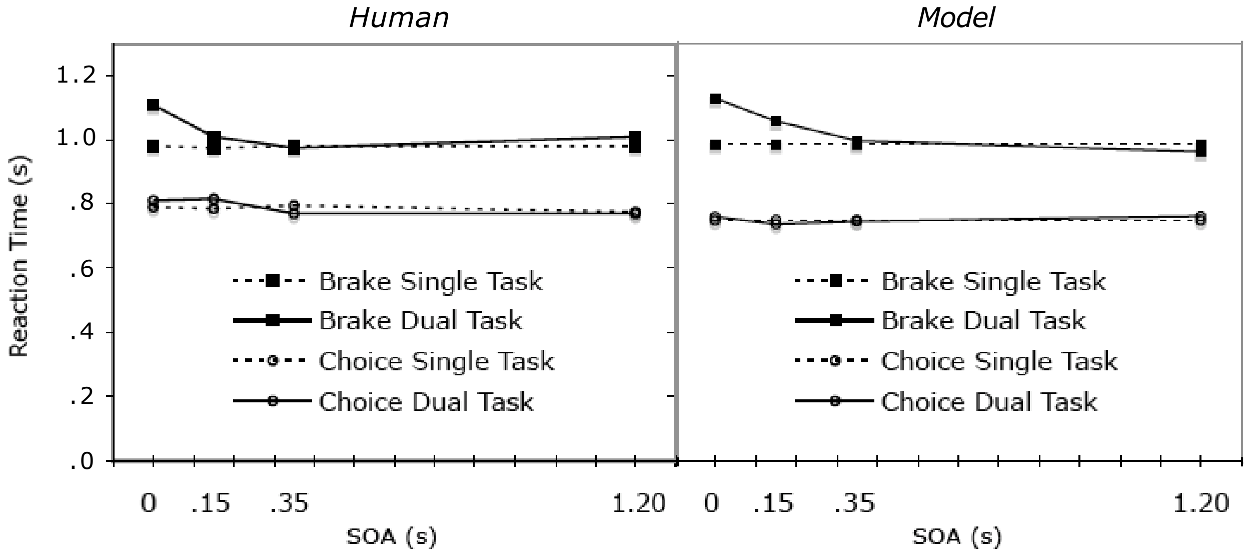

Driving
Choice
Procedural: Find road near point
Procedural: Find road far point
Procedural: Steer & accelerate
Procedural: Note vehicle is stable
(waiting until procedural is free)
Procedural: Find road near point
Procedural: Find road far point
(waiting until procedural is free)
Procedural: Steer & accelerate
1
(waiting until procedural is free)
Procedural: Note vehicle is stable
Procedural: Find road near point
Procedural: Find road far point
Procedural: Steer & accelerate
2
Aural: Detect tone
Procedural: Wait for next tone
(waiting 100 ms)
Procedural: Detect next tone
Aural: Detect tone
Procedural: Respond to stimulus
Manual: Press key
= Manual: Stee r; Pedal: Acc/Decelerate ; Visual: Encode far point object
★ = Lead-car brake lights appear (SOA = 150 ms)
1 = Brake lights noted during visual encoding
2 = Model overrides accelerator, produces brake response

Figure 13.

(a) reaction time by SOA



(b) reaction time by input/output modality

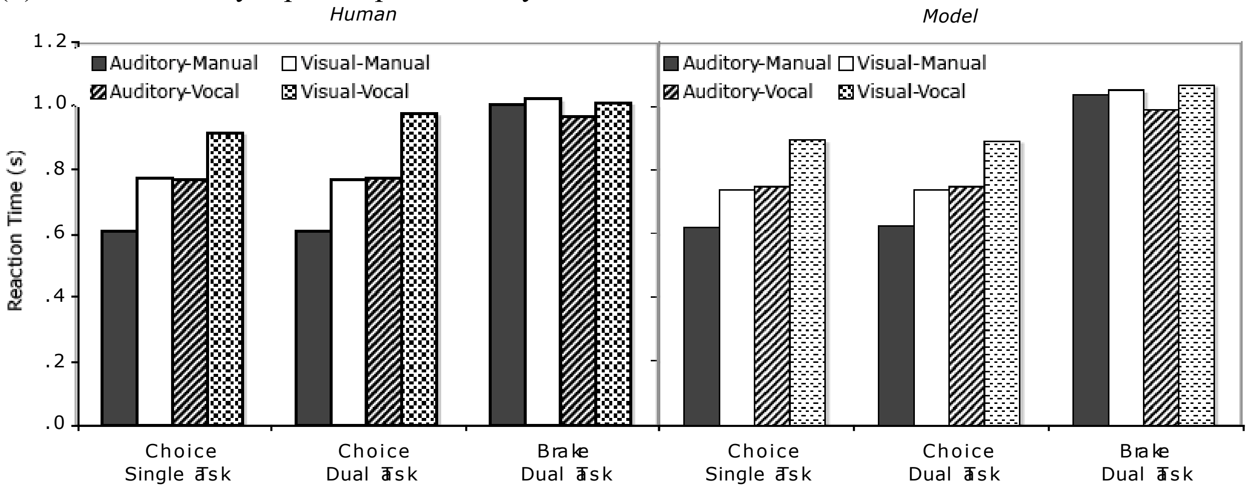


Figure 14.

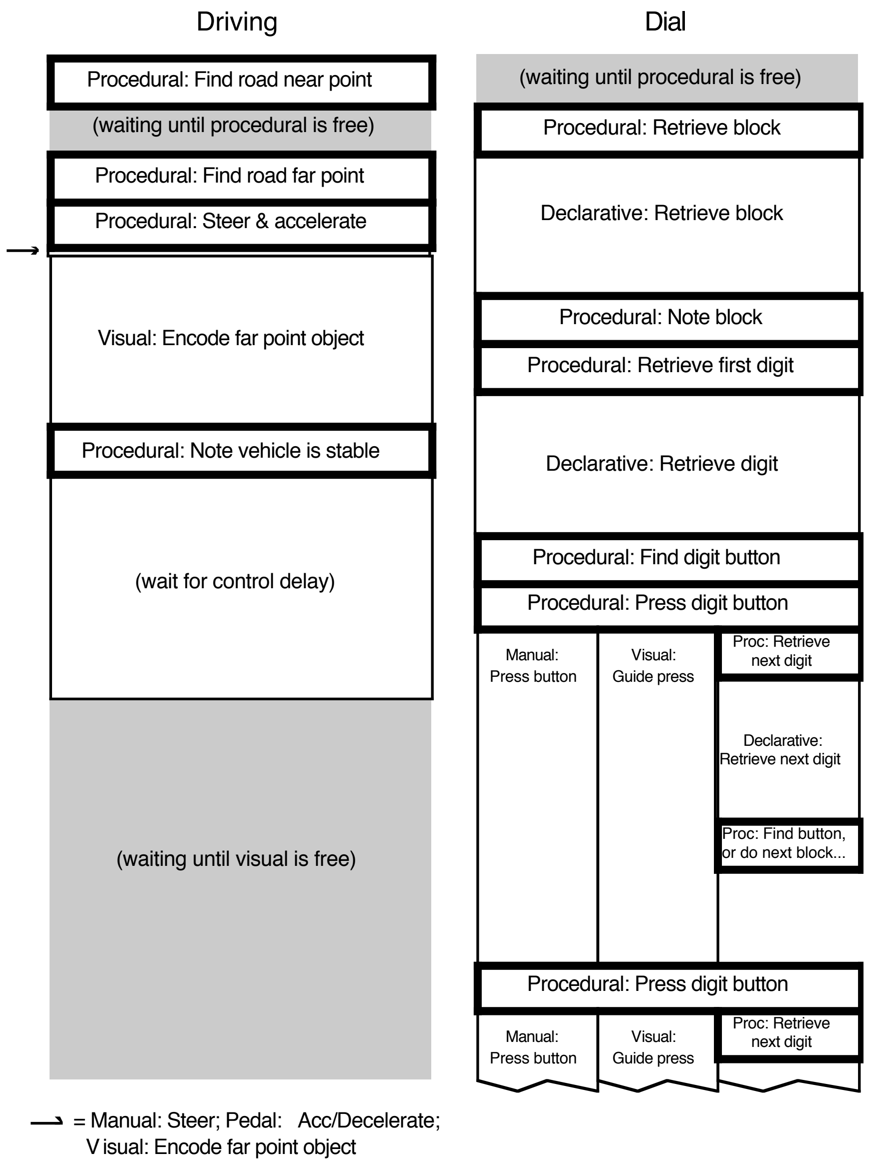
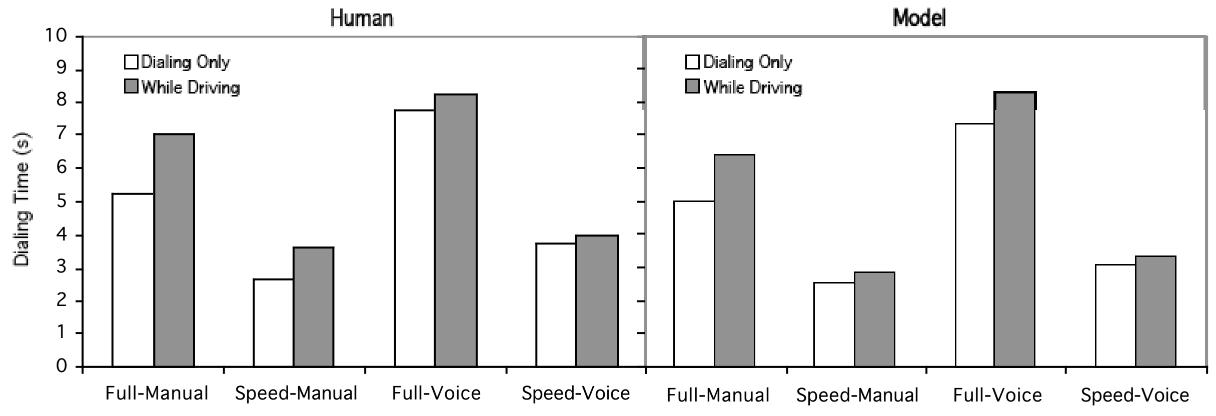

Driving
Dial
Procedural: Find road near point
(waiting until procedural is free)
Procedural: Find road far point
Procedural: Steer & accelerate
Visual: Encode far point object
Procedural: Note vehicle is stable
(wait for control delay)
(waiting until visual is free)
(waiting until procedural is free)
Procedural: Retrieve block
Declarative: Retrieve block
Procedural: Note block
Procedural: Retrieve first digit
Declarative: Retrieve digit
Procedural: Find digit button
Procedural: Press digit button
Manual: Press button
Visual: Guide press
Proc: Retrieve next digit
Declarative: Retrieve next digit
Proc: Find button, or do next block...
Procedural: Press digit button
Manual: Press button
Visual: Guide press
Proc: Retrieve next digit
= Manual: Steer; Pedal: Acc/Decelerate; Visual: Encode far point object

Figure 15.

(a) dialing time



(b) lateral velocity

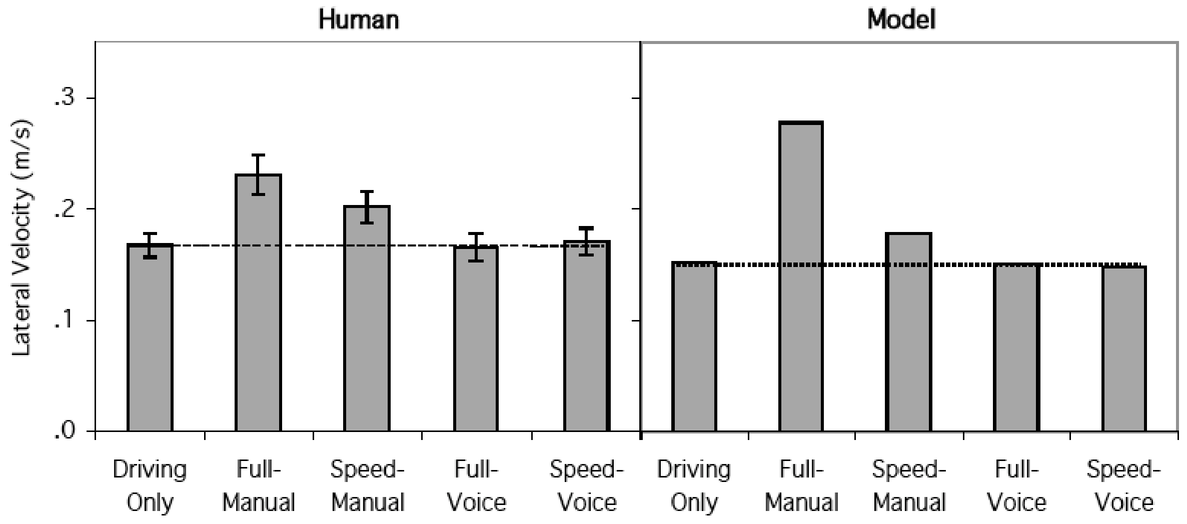
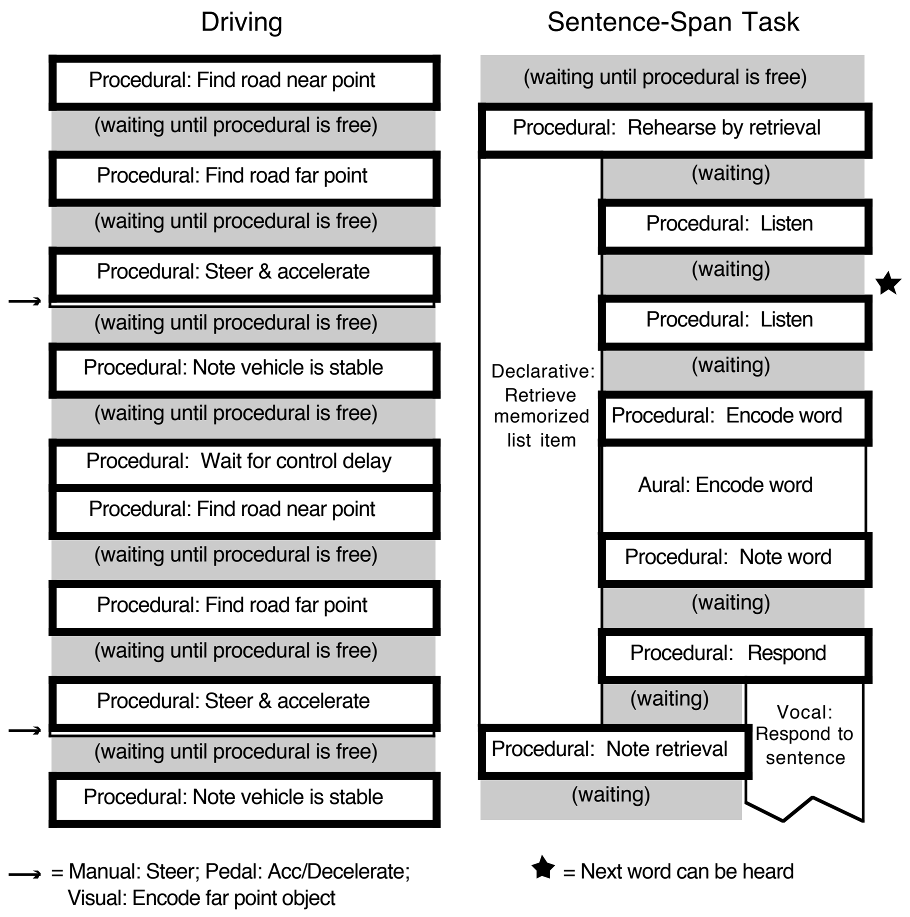


Figure 16.


Driving
Procedural: Find road near point
(waiting until procedural is free)
Procedural: Find road far point
(waiting until procedural is free)
Procedural: Steer & accelerate
(waiting until procedural is free)
Procedural: Note vehicle is stable
(waiting until procedural is free)
Procedural: Wait for control delay
Procedural: Find road near point
(waiting until procedural is free)
Procedural: Find road far point
(waiting until procedural is free)
Procedural: Steer & accelerate
(waiting until procedural is free)
Procedural: Note vehicle is stable
Sentence-Span Task
(waiting until procedural is free)
Procedural: Rehearse by retrieval
(waiting)
Procedural: Listen
(waiting)
Procedural: Listen
(waiting)
Declarative: Retrieve memorized list item
Procedural: Encode word
Aural: Encode word
Procedural: Note word
(waiting)
Procedural: Respond
(waiting)
Vocal: Respond to sentence
Procedural: Note retrieval
(waiting)
= Manual: Steer; Pedal: Acc/Decelerate; Visual: Encode far point object
= Next word can be heard