Cognitive Architecture

Michael D. Byrne

Department of Psychology
Rice University
6100 Main St., MS-25
Houston, TX 77005
byrne@acm.org
+1 713-348-3770 voice
+1 713-348-5221 fax

> "Designing interactive computer systems to be efficient and easy to use is important so that people in our society may realize the potential benefits of computer-based tools... Although modern cognitive psychology contains a wealth of knowledge of human behavior, it is not a simple matter to bring this knowledge to bear on the practical problems of design—to build an applied psychology that includes theory, data, and knowledge." (Card, Moran, & Newell, 1983, p. vii).

1. Introduction

Integrating theory, data, and knowledge about cognitive psychology and human performance in a way that is useful for guiding design in HCI is still not a simple matter. However, there have been significant advances since Card, Moran, and Newell wrote the above passage. One of the key advances is the development of cognitive architecture, the subject of this chapter. The chapter will first consider the what it is to be cognitive architecture and why cognitive architecture is relevant for HCI. In order to detail the present state of cognitive architectures in HCI, it is important to consider some of the past use of cognitive architectures in HCI research. Then, four architectures actively in use in the research community (LICAI/CoLiDeS, Soar, EPIC, and ACT-R/PM) and their application to HCI will be examined. The chapter will conclude with a discussion of the future of cognitive architectures in HCI.

1.1 What Are Cognitive Architectures?

Most any dictionary will list several different definitions for the word "architecture." For example, dictionary.com lists among them "a style and method of design and construction," e.g., Byzantine architecture; "orderly arrangement of parts; structure," e.g., the architecture of a novel; and one from computer science: "the overall design or structure of a computer system." What, then would something have to be qualify as a "cognitive architecture"? It is something much more in the latter senses of the word "architecture," an attempt to describe the overall structure and arrangement of a very particular thing, the human cognitive system. A cognitive architecture is a broad theory of human cognition based on a wide selection of human experimental data, and

implemented as a running computer simulation program. Young (Gray, Young, & Kirschenbaum, 1997; Ritter & Young, 2001) defines a cognitive architecture as an embodiment of "a scientific hypothesis about those aspects of human cognition that are relatively constant over time and relatively independent of task."

This idea has been a part of cognitive science since the early days of cognitive psychology and artificial intelligence, as manifested in the General Problem Solver or GPS (Newell & Simon, 1963), one of the first successful computational cognitive models. These theories have progressed a great deal since GPS, and are gradually becoming more and more broad. One of the best descriptions of the vision for this area is presented in Newell's (1990) book Unified Theories of Cognition. In it, Newell argues that the time has come for cognitive psychology to stop collecting disconnected empirical phenomena and begin seriously considering theoretical unification in the form of computer simulation models. Cognitive architectures are attempts to do just this.

Cognitive architectures are distinct from engineering approaches to artificial intelligence, which strive to construct intelligent computer systems by whatever technologies best serve that purpose. Cognitive architectures are designed to simulate human intelligence in a humanlike way (Newell, 1990). For example, the chess program that defeated Kasparov, Deep Blue, would not qualify as a cognitive architecture, because it does not solve the problem (chess) in a human-like way. Deep Blue uses massive search of the game space, while human experts generally look only a few moves ahead, but concentrate effectively on quality moves.

Cognitive architectures differ from traditional research in psychology in that work on cognitive architecture is integrative. That is, they include attention, memory, problem solving, decision making, learning, and so on. Most theorizing in psychology follows a divide-and-conquer strategy that tends to generate highly specific theories of a very limited range of phenomena; this has changed little since the 1970's (Newell, 1973). This limits the usefulness of such theories for an applied domain like HCI where users employ a wide range of cognitive capabilities in even simple tasks. Instead of asking "how can we describe this isolated phenomenon?" people working with

cognitive architectures can ask “how does this phenomenon fit in with what we already know about other aspects of cognition?”

Another important feature of cognitive architectures is that they specify only the human “virtual machine,” the fixed architecture. A cognitive architecture alone cannot do anything. Generally, the architecture has to be supplied with the knowledge needed to perform a particular task. The combination of an architecture and a particular set of knowledge is generally referred to as a model. In general, it is possible to construct more than one model for any particular task. The specific knowledge incorporated into a particular model is determined by the modeler. Because the relevant knowledge must be supplied to the architecture, the knowledge engineering task facing modelers attempting to model performance on complex tasks can be formidable.

Another centrally important feature of cognitive architectures is that they are software artifacts constructed by human programmers. This has a number of relevant ramifications. First, a model of a task constructed in a cognitive architecture is runnable and produces a sequence of behaviors. These behavior sequences can be compared with the sequences produced by human users to help assess the quality of a particular model. They may also provide insight into alternate ways to perform a task; that is, they may show possible strategies that are not actually utilized by the people performing the task. This can be useful in guiding interface design as well. Another feature of many architectures is that they enable the creation of quantitative models. For instance, the model may say more than just “click on button A and then menu B,” but may include the time between the two clicks as well. Models based on cognitive architectures can produce execution times, error rates, and even learning curves. This is a major strength of cognitive architectures as an approach to certain kinds of HCI problems and will be discussed in more detail in the next section.

On the other hand, cognitive architectures are large software systems, which are often considered difficult to construct and maintain. Individual models are also essentially programs, written in the “language” of the cognitive architecture. Thus, individual modelers need to have solid programming skills.

Finally, cognitive architectures are not in wide use among HCI practitioners. Right now, they exist primarily in academic research laboratories. One of the barriers for practitioners is that learning and using most cognitive architectures is itself generally a difficult task. However, this is gradually changing and some of the issues being addressed in this regard will be discussed in section 4. Furthermore, even if cognitive architectures are not in wide use by practitioners, this does not mean that they are irrelevant to practitioners. The next section highlights why cognitive architectures are relevant to a wide HCI audience.

1.2 Relevance to Human-computer Interaction

For some readers, the relevance of models that produce quantitative predictions about human performance will be obvious. For others, this may be less immediately clear. Cognitive architectures are relevant to usability as an engineering discipline, have several HCI-relevant applications in computing systems, and serve an important role in HCI as a theoretical science.

At nearly all HCI-oriented conferences, and many online resources, there are areas where corporations recruit HCI professionals. A common job title in these forums is “usability engineer.” Implicit in this title is the view that usability is, at least in part, an engineering enterprise. And while people with this job title are certainly involved in product design, there is a sense in which most usability engineering would not be recognized as engineering by people trained in more traditional engineering disciplines such as electrical or aerospace engineering. In traditional engineering disciplines, design is generally guided at least in part by quantitative theory. Engineers have at their disposal hard theories of the domain in which they work, and these theories allow them to derive quantitative predictions. Consider an aerospace engineer designing a wing. Like a usability engineer, the aerospace engineer will not start with nothing; a pre-existing design often provides a starting point. But when the aerospace engineer decides to make a change in that design, there is usually quantitative guidance about how the performance of the wing will change as a result of the change in design. This guidance, while quantitative, is not infallible, hence the need for evaluation tools like wind tunnels. This is not unlike the usability engineer’s usability test. However, unlike in usability testing, the aerospace has some quantitative idea about what the outcome of the test will

be, and this is not guided simply by intuition and experience, but by a quantitative theory of aerodynamics. In fact, this theory is now so advanced that few wind tunnels are being built anymore. Instead, they are being replaced by computer simulations, an outcome of the application of computational techniques to complex problems in aerodynamics called “computational fluid dynamics.” This has not entirely replaced wind tunnels, but the demand for wind tunnel time has clearly been affected by this development.

For the most part, the usability engineer lacks the quantitative tools available to the aerospace engineer. Every design must be subjected to its own wind tunnel (usability) test, and the engineer has little guidance about what to expect other than from intuition and experience with similar tests. While intuition and experience can certainly be valuable guides, they often fall short of more “hard” quantitative methods. Perhaps the engineer can intuit that interface “X” will allow users to complete tasks faster than with interface “Y,” but how much faster? 10%? 20%? Even small savings in execution times can add up to large financial savings for organizations when one considers the scale of the activity. The paradigm example is the telephone operators studied Gray, John, and Atwood (1993), where even a second saved on an average call would save the telephone company millions of dollars.

Computational models based on cognitive architectures have the potential to provide detailed quantitative answers, and for more than just execution times. Error rates, transfer of knowledge, learning rates, and other kinds of performance measures are all metrics than can often be provided by architecture-based models. Even if such models are not always precisely accurate in an absolute sense, they may still be useful in a comparative sense. For example, if a usability engineer is comparing interface A with interface B and the model at his or her disposal does not accurately predict the absolute times to complete some set of benchmark tasks, it may still accurately capture the difference between the two interfaces, which may be more than enough.

Additionally, there are certain circumstances when usability tests are impractical or prohibitively costly, or both. For example, access to certain populations such as physicians or astronauts may be difficult or expensive, so bringing them in for repeated usability tests may not be

feasible. While developing a model of a pilot or an air traffic controller performing an expert task with specialized systems may be difficult at first, re-running that model to assess a change made to the user interface should be much more straightforward than performing a new usability test for each iteration of the system. This is possible only with a quantitatively realistic model of the human in the loop, one that can produce things like execution times and error rates. Computational models can, in principle, act as surrogate users in usability testing, even for special populations.

Of course, some of these measures can be obtained through other methods such as GOMS analysis (Card, Moran, and Newell, 1983; John & Kieras, 1996) or cognitive walkthrough (Polson, Lewis, Reiman, & Wharton, 1992). However, these techniques were originally grounded in the same ideas as some prominent cognitive architectures and are essentially abstractions of the relevant architectures for particular HCI purposes. Also, architecture-based computational models provide things that GOMS models and cognitive walkthroughs do not. First, models are executable and generative. A GOMS analysis, on the other hand, is a description of the procedural knowledge the user has to have and the sequence of actions that must be performed to accomplish a specific task instance, while the equivalent computational model actually generates the behaviors, often in real time or faster. Equally importantly, computational models have the capacity to be reactive in real time. So, while it may be possible to construct a GOMS model which describes the knowledge necessary and the time it will take an operator to classify a new object on an air traffic controller's screen, a paper-and-pencil GOMS model cannot actually execute the procedure in response to the appearance of such an object. A running computational model, on the other hand, can.

Because of this property, architecture-based computational models have some other important uses beyond acting as virtual users in usability tests. One such use is in intelligent tutoring systems (ITSs). Consider the Lisp tutor (Anderson, Conrad, & Corbett, 1989). This tutoring system contained an architecture-based running computational model of the knowledge necessary to implement the relevant Lisp functions, and a module for assessing which pieces of this knowledge were mastered by the student. Because the model was executable, it could predict what action the student would take if the student had correct knowledge of how to solve the problem.

When the student took a different action, this told the ITS that the student was missing one or more relevant pieces of knowledge. The student could then be given feedback about what knowledge is missing or incomplete, and problems which exercise this knowledge could be selected by the ITS. By identifying students' knowledge, and the gaps in that knowledge, it is possible to generate more effective educational experiences. Problems which contain knowledge the student has already mastered can be avoided, to not bore the student with things they already know. This frees up the student to concentrate on the material they have not yet mastered, resulting in improved learning (Anderson, et al., 1989). While the Lisp tutor is an old research system, ITSs based on the same underlying cognitive architecture with the same essential methodology have been developed for more pressing educational needs such as algebra and geometry and are now sold commercially (see www.carnegielearning.com).

There is another HCI-relevant application for high-fidelity cognitive models: populating simulated worlds or situations. For example, training an F-16 fighter pilot is expensive, even in a simulator, because that trainee needs to face realistic opposition. Realistic opposition consists of other trained pilots, so training one person requires taking several trained pilots away from their normal duties (i.e., flying airplanes on real missions). This is difficult and expensive. If, however, the other pilots could be simulated realistically, then the trainee could face opposition that would have useful training value, without having to remove already-trained pilots from their duties. There are many training situations like this, where the only way to train someone is to involve multiple human experts who must all be taken away from their regular jobs. However, the need for expensive experts can potentially be eliminated (or at least reduced) by using architecturally-based cognitive models in place of the human experts. The U.S. military has already started to experiment with just such a scenario (Jones, Laird, Nielsen, Coulter, Kenny, & Koss, 1999). There are other domains besides training where having realistic opponents is desirable, such as video games. Besides things like texture-mapped 3D graphics, one of the features often used to sell games is network play. This enables players to engage opponents whose capabilities are more comparable to their own than typical computer-generated opponents. However, even with network play, it is not always possible

for a game to find an appropriate opponent. If the computer generated opponent were a more high-fidelity simulation of a human in terms of cognitive and perceptual-motor capabilities, then video game players would have no difficulty finding appropriate opponents without relying on network play. While this might not be the most scientifically interesting use of cognitive architectures, it seems inevitable that cognitive architectures will be used in this way.

Cognitive architectures are also theoretically important to HCI as an interdisciplinary field. Many people (including some cognitive psychologists) find terms from cognitive psychology such as “working memory” or “mental model” vague and ill-defined. A harsh evaluation of explanations relying on such terms is found in Salthouse (1988, p. 3): “It is quite possible that interpretations relying on such nebulous constructs are only masquerading ignorance in what is essentially vacuous terminology.” Computational cognitive architectures, on the other hand, require explicit specifications of the semantics of theoretical terms. Even if the architectures are imperfect descriptions of the human cognitive system, they are at a minimum well-specified and therefore clearer in what they predict than strictly verbal theories.

A second theoretical advantage of computational theories such as cognitive architectures is they provide a window into how the theory actually works. As theories grow in size and number of mechanisms, the interactions of those mechanisms becomes increasingly difficult to predict analytically. Computer simulations permit relatively rapid evaluations of complex mechanisms and their interactions. (For an excellent discussion of this topic, see Simon, 1996.) Another problem with theories based solely on verbal descriptions is that it can be very difficult to assess the internal coherence of such theories, while such assessment is much more straightforward with computational models. Verbal theories can easily hide subtle (and not so subtle) inconsistencies that make them poor scientific theories. Computational models, on the other hand, force explanations to have a high level of internal coherence; theories that are not internally consistent are typically impossible to implement on real machines.

Finally, HCI is an interdisciplinary field, and thus theories that are fundamentally interdisciplinary in nature are appropriate. Cognitive architectures are such theories, combining

computational methods and knowledge from the artificial intelligence end of computer science with data and theories from cognitive psychology. While cognitive psychology and computer science are certainly not the only disciplines that participate in HCI, they are two highly visible forces in the field. Psychological theories that are manifested as executable programs should be less alien to people with a computer science background than more traditional psychological theories.

Thus, cognitive architectures are clearly relevant to HCI at a number of levels. This fact has not gone unnoticed by the HCI research community. In fact, cognitive architectures have a long history in HCI, dating back to the original work of Card, Moran, and Newell (1983).

2. Brief Look at Past Systems in HCI

The total history of cognitive architectures and HCI would be far too long to document in a single chapter, however, it is possible to touch on some highlights. While not all of the systems described in this section qualify as complete cognitive architectures, they all share intellectual history with more current architectures and influenced their development and use in HCI. Finally, many of the concepts developed in these efforts are still central parts of the ongoing research on cognitive architecture. Also, there is a natural starting point:

2.1 The Model Human Processor (MHP) and GOMS

The Psychology of Human-Computer Interaction (Card, Moran, and Newell, 1983) is clearly a seminal work in HCI, one of the defining academic works in the early days of the field. While that work did not produce a running cognitive architecture, it was clearly in the spirit of cognitive architectures and was quite influential in the development of current cognitive architectures. Two particular pieces of that work are relevant here, the Model Human Processor (MHP) and GOMS.

The MHP represents a synthesis of the literature on cognitive psychology and human performance up to that time, and sketches the framework around which a cognitive architecture could be implemented. The MHP is a system with multiple memories and multiple processors, and many of the properties of those processors and memories is described in some detail. (See Figure

1.) Card, Moran, and Newell also specified the interconnections of the processors and a number of general operating principles. In this system, there are three processors: one cognitive, one perceptual, and one motor. In some cases the system behaves essentially serially. For instance, in order for the system to press a key in response to the appearance of a light, the perceptual processor must detect the appearance of the light and transmit this information to the cognitive processor. The cognitive processor's job is to decide what the appropriate response should be, and then transmit that to the motor processor, which is responsible for actually executing the appropriate motor command. In this situation, the processors act serially, one after another. However, in more complex tasks such as transcription typing, all three processors will often be working in parallel.

---------- insert Figure 1 about here ----------

Besides the specification of the timing for each processor and the connectivity of the processors, Card, Moran, and Newell lay out some general operating principles ranging from very general and qualitative to detailed and quantitative. For example, Principle P9, the Problem Space Principle, states:

The rational activity in which people engage to solve a problem can be described in terms of (1) a set of states of knowledge, (2) operators for changing one state into another, (3) constraints on applying operators, and (4) control knowledge for deciding which operator to apply next. (Card, et al. 1983, p. 27)

This is a particularly general and somewhat vague principle. In contrast, consider Principle P5, Fitts's Law:

The time T_{pos} to move the hand to a target of size S which lies a distance D away is given by:

$$T_{pos} = I_M \log_2(D / S + .5)$$

where I_M = 100 [70~120] ms/bit. (Card, et al. 1983, p. 27)

This is a very specific principle which quantitatively describes hand movement behavior, which is highly relevant to, say, pointing with a mouse. Overall, the specification of the MHP is quite thorough, and lays out a basis for a cognitive architecture able to do a wide variety of HCI-

relevant tasks. However, Card, et al. did not implement the MHP as a running cognitive architecture. This is likely for pedagogical reasons; it is not necessary to have a complete, running cognitive architecture for the general properties of that architecture to be useful for guiding HCI researchers and practitioners. At the time, computational modeling was the domain of a very specialized few in cognitive psychology.

Card, et al. lays out another concept that has been highly influential throughout HCI and particularly in the community of computational modelers. This is GOMS, which stands for goals, operators, methods, and selection rules. GOMS is a framework for task analysis that describes routine cognitive skills in terms of the listed four components. Routine cognitive skills are those where the user knows what the task is and how to do the task without doing any problem solving. Text editing with a familiar editor is the prototypical case of this, but clearly a great many tasks in of interest in HCI could be classified as routine cognitive skills. Thus, the potential applicability of GOMS is quite broad. Indeed, GOMS has been applied to a variety of tasks; the Web site www.gomsmodel.org lists 143 GOMS-related papers in its bibliography.

What does a GOMS analysis provide? Essentially, a GOMS analysis of a task describes the hierarchical procedural knowledge a person must have to successfully complete that task. Based on that, and the sequence of operators that must be executed, it is possible to make quantitative predictions about the execution time for a particular task. Other analyses, such as predictions of error, functionality coverage, and learning time are also sometimes possible. Since the original formulation presented in Card, et al., a number of different forms of GOMS analysis have been developed, each with slightly different strengths and weaknesses (John & Kieras, 1996).

The core point as it relates to cognitive architectures is that GOMS analysis is originally based on a production rule analysis (Card, personal communication, 1999). Because this will come up several times, a brief introduction to production systems is warranted. Production rules are IF-THEN condition-action pairs, and a set of production rules (or simply “productions” or just “rules”) and a computational engine that interprets those productions is called a production system. In addition to productions, production systems contain some representation of the current state.

This representation typically consists of a set of loosely-structured data elements such as propositions or attribute-value pairs. This set is called the “working memory” or “declarative memory.” Because “working memory” is also a psychological term with somewhat different meaning in that literature, “declarative memory” will be used in all further discussions.

The operation of a production system is cyclic. On each cycle, the system first goes through a pattern-matching process. The IF side of each production tests for the presence of a particular pattern in declarative memory. When the IF conditions of a production are met, the production is said to fire and the actions specified on the THEN side are executed. The actions can be things like pressing a button or even some higher-level abstraction of action (e.g., “turn left”). Actions also include modifying the contents of declarative memory, which usually means that a different production or productions will match on the next cycle. At this abstract and purely symbolic level, production systems are Turing complete and thus can compute anything that is computable (Newell, 1990), thus, they should be flexible enough to model the wide array of computations performed by the human cognitive system.

This is relevant to cognitive architectures because most cognitive architectures are (or contain) production systems. GOMS was actually abstracted from production rule analysis. Card, et al. discovered that, for routine cognitive skills, the structure of the productions was quite similar across tasks and a more abstract representation was possible. This representation is the original GOMS formulation. Thus, translating a GOMS analysis into production rules, the language of most cognitive architectures, is generally straightforward. Similarly, for routine cognitive skills, it is often relatively simple to derive a GOMS analysis from the set of productions used to model the task. Models based on cognitive architectures can go well beyond routine cognitive skills, but this connection has certainly influenced the evolution of research on cognitive architecture and HCI. This connection has also fed back into research and development of GOMS techniques themselves, such as NGOMSL (Kieras, 1988). NGOMSL (Natural GOMS Language) allows the prediction of learning time for the knowledge described in a GOMS model based on a theory of transfer of

training referred to as cognitive complexity theory (CCT), which will be described in more detail in the next section.

2.2 Cognitive Complexity Theory (CCT)

When someone has learned to perform a task with a particular interface and must switch, doing the same task with a new interface, how much better off will they be than someone just learning to do the task with the new interface? That is, how much is the knowledge gained from using the old interface "transferred" to using the new interface? This question has intrigued psychologists for at least a century, and having some answers to this question has implications for training programs, user interface design, and many other areas. Cognitive complexity theory (Bovair, Kieras, & Polson, 1990; Kieras & Polson, 1985) is a psychological theory of transfer of training applied to HCI. Most relevant to the current discussion, this theory is based on production rules. The major points of CCT are as follows:

• Knowledge of the procedures that people need to execute to perform routine tasks can be represented with production rules. The relevant production rules can be generated based on a GOMS analysis of the task to be modeled.

• The complexity of a task will be reflected in the number and content of the production rules. When certain conventions are adopted about the style of those production rules, complexity is reflected almost entirely in the number of rules.

• The time it takes to execute a procedure can be predicted with a production system that interprets those rules along with a set of typical operator times, e.g. the time it takes to type a three-letter command. The production interpreter used in this work was not intended to be a general cognitive architecture, but the production system framework is certainly consistent with current architectures.

• The time it takes to learn a task is a function of the number of new rules that the user must learn. "New" is clearly defined in this context. If the user already has a production, and a new task requires a rule that is similar (again, similarity is well-defined based on the production rule syntax), then the rule for the new task need not be learned.

• Some predictions about errors and speedup with practice can also be gleaned from the contents of the production rules.

Obviously, this was an ambitious agenda and there are many subtleties. For example, the notion of a "task" as the term was used in the description of CCT actually includes more than just the task at an abstract level. Consider a simple instance of a text-editing task, deleting the word "redux" from the middle of a sentence. The actual commands needed to accomplish this task could be very different in different text editors, thus, modeling the "delete word" task would require two different sets of productions, one for each editor. That is, the necessary knowledge, and thus the production rules for representing it, are actually a function both of the task from the user point of view (e.g. "delete word") and the interface provided to accomplish the task. Transfer from one text editor to another therefore depends a great deal on the particulars of each interface. CCT thus predicts asymmetrical transfer: learning editor A after editor B should not be the same as learning editor B after editor A.

CCT models, like a GOMS analysis, omit modeling many details of user behavior. In general, anything that falls outside the domain of procedural knowledge (how-to-do-it knowledge) is not modeled. This means that the model does not model motor actions such as keypresses, and instead has a "DoKeystroke" primitive operator. Nor do CCT models model things like natural language comprehension, clearly a requirement in text editing. CCT models also do not include any model of the perceptual processes required by users--the model was simply given information about the state of the display, and did not have to, for example, look to see if the cursor was over a particular character. This is the same scope as a typical GOMS model, though a CCT model is more formalized and quantitative than the GOMS models described by Card, et al. (1983).

In spite of these limitations (or perhaps in part because these limitations allowed the researchers to concentrate on the most central aspects of the phenomena), CCT fared very well. Numerous laboratory experiments provide empirical support for many of the claims of CCT (see especially Bovair, Kieras, & Polson, 1990). The CCT framework was developed and validated in greatest detail to pre-GUI text editing, but it has also been applied to menu-based systems (Polson,

Muncher, & Engelbeck, 1986) and a control panel device (Kieras & Bovair, 1986). Singley and Anderson (1989) provide a strikingly similar analysis of transfer of training as well as supporting empirical results, lending credence to the CCT analysis. CCT was certainly one of the most prominent early successes of computational modeling in HCI.

2.3 CAPS

CAPS (collaborative activation-based production system; Just & Carpenter, 1992) is a cognitive architecture designed to model individual differences in working memory (WM) capacity and the effects of working memory load. This speciality is applicable to a number of HCI situations. Certainly, some kinds of user interfaces can create excessive working memory demands, for example, phone-based interfaces. In phone-based interaction (PBI), options do not remain on a screen or in any kind of available storage; rather, users are forced to remember the options presented. This seems like a prime candidate for modeling with a system designed to capture the effects of working memory demand, and this is exactly what Huguenard, Lerch, Junker, Patz, and Kass (1997) did. Their data showed that, contrary to guideline advice and most people's intuition, restricting phone menus to only a few (three) items each does not reduce error rates. The CAPS-based model provided a clear theoretical account of this phenomenon. The model showed that short menus are not necessarily better in PBI because of two side-effects of designing menu hierarchies with few options at each level. First, for the same number of total items, this increases menu depth, which creates working memory demand. Second, with fewer items at each level, each individual item has to be more general and therefore more vague, especially at the top levels of the hierarchy. This forces users to spend WM resources on disambiguating menu items when they are in a situation where WM demands outstrip supply.

Another application of CAPS that is HCI-relevant is the account of postcompletion error provided by Byrne and Bovair (1997). What is a postcompletion error? Anecdotal evidence and intuition suggests that, when interacting with man-made artifacts, certain kinds of errors occur with greater frequency than others. In particular, there is an entire family of errors that seem intuitively common, these are errors that people make when there is some part of a task that occurs after the

main goal of the task has been accomplished (hence “postcompletion”). Nearly everyone reports having made an error of this type at one time or another. Here are two prototypical examples:

• Leaving the original on the glass of a photocopier. The main goal one generally has when using a photocopier is “get copies” and this goal is satisfied before one remove the original document. This error is less common now that many photocopiers include document feeders; the more current equivalent is leaving a document on the glass in a flatbed scanner.

• Leaving one’s bank card in an automated teller machine (ATM). Again, the main goal is something on the order of “get cash,” and in many ATMs card removal occurs after the cash is dispensed. This error was common enough in the first generation of ATMs that many ATMs are now designed in such a way that this error is now impossible to make.

There are many others, such as leaving the gas cap off after filling up the car’s gas tank, leaving change in vending machines, and more—most readers can probably think of several others. While numerous HCI researchers were aware of this class of error (e.g. Young, Barnard, Simon, & Whittington, 1989; Polson, et al., 1994), no account had previously been developed which explained why this type of error is persistent, yet not so frequent that it occurs every time. The CAPS model provides just such an account, and can serve as a useful example of the application of a cognitive architecture to an HCI problem.

Like most other production systems, CAPS contains two kinds of knowledge, declarative memory and productions. Declarative memory elements in CAPS also have associated with them an activation value, and elements below a threshold level of activation cannot be matched by productions’ IF sides. Additionally, unlike most other production systems, the THEN side of a CAPS production may request that the activation of an element be incremented. For this to be truly useful in modeling working memory, there is a limit to the total amount of activation available across all elements. If the total activation exceeds this limit, then all elements lose some activation to bring the total back within the limit. This provides a mechanism for simulating human working memory limitations.

In Byrne and Bovair's postcompletion error model, there is a production which increments the activation of subgoals when the parent goal is active and unsatisfied. So, to use the photocopier example, the "get copies" subgoal supplies activation to all the unfulfilled subgoals throughout the task. However, when the "get copies" goal is satisfied, the activation supply to the subgoals stops. Because the goal to remove the original is a subgoal of that goal, it loses its activation supply. Thus, what the model predicts is that the postcompletion subgoals are especially vulnerable to working memory load, and lower-capacity individuals are more "at risk" than higher-capacity individuals. Byrne and Bovair conducted an experiment to test this prediction, and the data supported the model.

This is a nice demonstration of the power of cognitive architectures. Byrne and Bovair neither designed nor implemented the CAPS architecture, but were able to use the theory to construct a model that made empirically-testable predictions, and those predictions were borne out. While CAPS is unlikely to guide much future HCI work (its designers are no longer developing and supporting it because they have gone in a different direction), it provides an excellent example case.

3. Contemporary Architectures

There are currently cognitive architectures that are being actively developed, updated, and applied to HCI-oriented tasks. Three of the four most prominent are production systems, or rather are centrally built around production systems. These three are Soar, EPIC, and ACT-R (particularly ACT-R/PM). While all contain production rules, the level of granularity of an individual production rule varies considerably from architecture to architecture. Each one has a different history and each one has a unique focus. They all share a certain amount of intellectual history; in particular they have all been influenced one way or another by the MHP, and by each other. At some level they may have more similarities than differences, whether this is because they borrow from one another or because the science is converging is still an open question. The fourth system is somewhat different than these three production system models and will be considered first.

3.1 LICAI/CoLiDeS

LICAI (Kitajima & Polson, 1997) is a good example of a non-production system architecture and has been used in an HCI context. All the work discussed up to this point more or less assumes that the users being modeled are relatively skilled with the specific interface being used; these approaches do a poor job of modeling relatively raw novices. One of the main goals of LICAI is addressing this concern. The paradigm question addressed by LICAI is "how do users explore a new interface?"

Unlike the other architectures discussed, LICAI's central control mechanisms are not based on a production system. Instead, LICAI is built around an architecture originally designed to model human discourse comprehension, construction-integration (C-I; Kintsch, 1998). Like production systems, C-I's operation is cyclic. However, what happens on those cycles is somewhat different than what happens in a production system. Each cycle is divided into two phases, construction and integration (hence the name). In the construction phase, an initial input (e.g., the contents of the current display) is fed into a weakly-constrained rule-based process which generates a network of propositions. Items in the network are linked on the basis of their argument overlap. For example, the goal of "graph data" might be represented with the proposition (PERFORM GRAPH DATA). Any proposition containing GRAPH or DATA would thus be linked to that goal.

Once the construction phase completes, the system is left with a linked network of propositions. What follows is the integration phase, in which activation propagates through the network in a neural network-like fashion. Essentially, this phase is a constraint-satisfaction phase, which is used to select one of the propositions in the network as the "preferred" one. For example, the system may need to select the next action to perform while using an interface. Action representations will be added to the network during the construction phase, and an action will be selected during the integration phase. The action will be performed, and the next cycle initiated. Various C-I models have used this basic process to select things other than actions. The original C-I system used these cycles to select between different interpretations of sentences.

There are three main kinds of cycles in LICAI: one type selects actions, one generates goals, and one selects goals. This is in contrast to how most HCI tasks have been modeled in

production system architectures; in such systems, the goals are usually included in the knowledge given to the system. This is not true in LICAI, in fact, the knowledge given to LICAI by the modelers is quite minimal. For the particular application of LICAI, which was modeling users who knew how to use a Macintosh for other tasks (e.g. word processing) and were now being asked to plot some data using a Macintosh program called CricketGraph (one group of users actually worked with Microsoft Excel), it included some very basic knowledge about the Macintosh GUI and some knowledge about graphing. Rather than supply the model with the goal hierarchy, Kitajima and Polson gave the model the same somewhat minimal instructions as the subjects. One of the major jobs of the LICAI model, then, was to generate the appropriate goals as they arose while attempting to carry out the instructions.

Again, this illustrates one of the strengths of using a cognitive architecture to model HCI tasks. Kitajima and Polson did not have to develop a theory of text comprehension for LICAI to be able to comprehend the instructions given to subjects, since LICAI is based on an architecture originally designed to do text comprehension, they essentially got that functionality gratis. Additionally, they did not include

just any text comprehension engine, but one that makes empirically-validated predictions about how people represent the text they read. Thus, the claim that the model started out with roughly the same knowledge as the users is highly credible.

The actual behavior of the model is also revealing, as it exhibits many of the same exploratory behaviors as the users. First, the model pursues a general strategy that can be classified as label-following (Polson, et al., 1994). The model, like the users, had a strong tendency to examine anything on the screen that had a label matching, or nearly matching (i.e., a near synonym) a key word in the task instructions. When the particular subtask being pursued by the model contained steps which were well-labeled, the users were rapid, which the model predicted. While this prediction is not counter-intuitive, it is important to note that LICAI is not programmed with this strategy. This strategy naturally emerges through the normal operation of construction-integration through the linkages created by shared arguments. The perhaps less intuitive result—

modeled successfully by LICAI—is the effect of the number of screen objects. During exploration in this task, users were slower to make choices if there were more objects on the screen, but only if those items all had what were classified as poor labels. In the presence of good labels (literal match or near synonym), the number of objects on the screen did not affect decision times, for the users or for LICAI.

The programmers who implemented the programs operated by the users put in several clever direct manipulation tricks. For example, to change the properties of a graph axis, one double-clicks on the axis and a dialog box specifying the properties of that axis appears. Microsoft Excel has some functionality that is most easily accessed by drag-and-drop. Fanzke (1994) found that in a majority of first encounters with these kind of direct manipulations, users required hints from the experimenter to be able to continue, even after 2 minutes of exploration. LICAI also fails at these interactions because there are no appropriate links formed between any kind of task goal and these unlabeled, apparently static screen objects during the construction phase. Thus, these screen objects tend to receive little activation during the integration phase, and actions involving other objects are always selected.

Overall, LICAI does an excellent job of capturing many of the other empirical regularities in exploratory learning of a GUI interface. This is an important issue for many interfaces, particularly any interface that is aimed at a walk-up-and-use audience. While currently common walk-up-and-use interfaces, such at ATMs, provide simple enough functionality that this is not always enormously difficult, this is not the case for more sophisticated systems, such as many information kiosks.

More recently, LICAI has been updated (and renamed to CoLiDeS, for Comprehension-based Linked model of Deliberate Search; Kitajima, Blackmon, & Polson, 2000) to handle interaction with Web pages. This involves goals that are considerably less well-elucidated and interfaces with a much wider range of semantic content. In order to help deal with these complexities, LICAI has been updated with a more robust attentional mechanism and a much more sophisticated notion of semantic similarity based on Latent Semantic Analysis (LSA; Landauer &

Dumais, 1994). CoLiDeS shows the effects of poor labels and poor hierarchical organization on Web navigation, getting "lost" in much the same way as real users. This is a promising tool for the analysis of semantically rich but functionality-poor domains such as the Web. Interestingly, Pirolli and Card (1999) have implemented a similar model in a modified version of ACT-R they term ACT-IF, where the IF stands for "information forager." Whether these systems will ultimately converge, diverge, or simply complement one another is an open question.

3.2 Soar

The development of Soar is generally credited to Allan Newell (especially Newell, 1990), and Soar has been used to model a wide variety of human cognitive activity from syllogistic reasoning (Polk & Newell, 1995) to flying combat aircraft in simulated wargames (Jones, et al., 1999). Soar was Newell's candidate "unified theory of cognition" and was the first computational theory to be offered as such.

While Soar is a production system, it is possible to think of Soar at a more abstract level. The guiding principle behind the design of Soar is Principle P9 from the Model Human Processor, the Problem Space Principle. Soar casts all cognitive activity as occurring in a problem space, which consists of a number of states. States are transformed through the application of operators. Consider Soar playing a simple game like tic-tac-toe as player X. The problem space is the set of all the states of the tic-tac-toe board—not a very large space. The operators available at any given state of that space are placing an X at any of the available open spaces on the board. Obviously, this is a simplified example; the problem space and the available operators for flying an F-16 in a simulated wargame are radically more complex.

Soar's operation is also cyclic, but the central cycle in Soar's operation is called a decision cycle. Essentially, on each decision cycle, Soar answers the question "what do I do next?" Soar does this in two phases. First, all productions that match the current contents of declarative memory fire. This usually causes changes in declarative memory, so other productions may now match. Those productions are allowed to fire, and this continues until no new productions fire. At this time, the decision procedure gets executed, in which Soar examines a special kind of declarative memory

element, the preference. Preferences are statements about possible actions, for example “operator o3 is better than o5 for the current operator” or “s10 rejected for supergoal state s7.” Soar examines the available preferences and selects an action. Thus, each decision cycle may contain many production cycles. When modeling human performance, the convention is that each decision cycle lasts 50 ms, so productions in Soar are very low-level, encapsulating knowledge at a very small grain size. This distinguishes Soar productions from those found in other production systems.

Other than the ubiquitous application of the problem space principle, Soar’s most defining characteristics come from two mechanisms developed specifically in Soar, universal subgoaling and a general-purpose learning mechanism. Because the latter depends on the former, universal subgoaling will be described first. One of the features of Soar’s decision process is that it is not guaranteed to have an unambiguous set of preferences to work with. Alternately, there may be no preferences listing an acceptable action. Perhaps the system does not know any acceptable operators for the current state, or perhaps the system lacks the knowledge of how to apply the best operator. Whatever the reason, if the decision procedure is unable to select an action, an impasse is said to occur. Rather than halting or entering some kind of failure state, Soar sets up a new state in a new problem space with the goal of resolving the impasse. For example, if multiple operators were proposed, the goal of the new problem space is to choose between the proposed operators. In the course of resolving one impasse, Soar may encounter another impasse and create another new problem space, and so on. As long as the system is provided with some fairly generic knowledge about resolving degenerate cases (e.g. if all else fails, choose randomly between the two good operators), this universal subgoaling allows Soar to continue even in cases where there is little knowledge.

Learning in Soar is a by-product of universal subgoaling. Whenever an impasse is resolved, Soar creates a new production rule. This rule summarizes the processing that went on in the substate. The resolution of an impasse makes a change to the superstate (the state in which the impasse originally occurred), this change is called a result. This result becomes the condition, or THEN, side of the new production. The condition, of IF, side of the production is generated through

a dependency analysis by looking at any declarative memory item matched in the course of determining this result. When Soar learns, it learns only new production rules, and it only learns as the result of resolving an impasse. It is important to realize that an impasse is not equated with failure or an inability to proceed in the problem-solving, but may arise simply because, for example, there are multiple good actions to take and Soar has to choose one of them. Soar impasses regularly when problem-solving and thus learning is pervasive in Soar.

Not surprisingly, Soar has been applied to a number of learning-oriented HCI situations. One of the best examples is the recent work by Altmann (Altmann & John, 1999; Altmann, 2001). Altmann collected approximately 80 minutes of data from an experienced programmer while she worked at understanding and updating a large computer program by examining a trace. These data included verbal protocols (i.e., thinking aloud) as well as a log of the actions taken (keypresses and scrolling). About once every three minutes, the programmer scrolled back to find a piece of information that had previously been displayed. Altmann constructed a Soar model of her activity. This model is a kind of comprehension model which attempts to gather information about its environment; it is not a complete model of the complex knowledge of an experienced programmer. The model attends to various pieces of the display, attempting to comprehend what it sees, and issues commands. Comprehension in this context is not the same as in C-I based models, but rather is manifested in this model as an attempt to retrieve information about the object being comprehended.

When an item is attended, this creates what Altmann termed an episodic trace, that is, a production that notes that the object was seen at a particular time. Because learning is pervasive, Soar creates many new rules like this. However, because of the dependency-based learning mechanism, these new productions are quite specific to the context in which the impasse originally occurred. Thus, the “index” into the model’s (fairly extensive) episodic memory consists of very specific cues, usually found on the display. Seeing a particular variable name is likely to trigger a memory for having previously seen that variable name. Importantly, this memory is generated automatically, without need for the model to deliberately set goals to remember particular items.

While Altmann and John (1999) is primarily a description of the model, Altmann (2001) discusses some of the HCI ramifications for this kind of always-on episodic memory trace, and discusses this in terms of display clutter. While avoiding display clutter is hardly new advice, it is generally argued that it should be avoided for visual reasons (e.g., Tullis, 1983). However, what Altmann's model shows is that display clutter can also have serious implications for effective use of episodic memory. Clutter can create enormous demands for retrieval. Since more objects will generally be attended on a cluttered display, the episodic trace will be large, lowering the predictive validity for any single cue. While this certainly seems like a reasonable account on the surface, it is unlikely that kind of analysis would have been generated if it had not been guided by a cognitive architecture which provided the omnipresent learning of Soar.

A second implication of Altmann's model is the surprising potential utility of browsing. Simple browsing creates an episodic trace of the objects encountered, regardless of intent, and thus browsing a complex interface may pay off later in the learning curve. Of course, for this to be most effective, the interface has to be structured correctly in order to best support retrieval of the episodic trace, and it is not clear exactly what the best organization would be to support such browsing.

Soar has also been used to implement models of exploratory learning, somewhat in the spirit of LICAI. There are two prominent models here, one called IDXL (Rieman, Young, & Howes, 1996) and a related model called Task-Action learner (Howes & Young, 1996). These models both attempt to learn unfamiliar GUI interfaces. IDXL operates in the same graphing domain as LICAI, while the Task-Action Leaner starts with even less knowledge and learns basic GUI operations such as how to open a file. For brevity, only IDXL will be described in detail.

IDXL goes through many of the same scanning processes as LICAI, but must rely on very different mechanisms for evaluation since Soar is fundamentally different than LICAI. IDXL models evaluation of various display elements as search through multiple problem spaces, one which is an internal search through Soar's internal knowledge and the other a search through the display. As items are evaluated in the search, Soar learns productions which summarize the products of each evaluation. At first, search is broad and shallow, with each item receiving a

minimum of elaboration. However, that prior elaboration guides the next round of elaboration, gradually allowing IDXL to focus in on the "best" items. This model suggests that a number of ways in which interface designers could thus help learners acquire a new the knowledge needed to utilize a new interface. Like the LICAI work, the IDXL work highlights the need for good labels to guide exploration. A more radical suggestion is based on one of the more subtle behavior of users and IDXL. When exploring and evaluating alternatives, long pauses often occur on particular menu items. During these long pauses, IDXL is attempting to determine the outcome of selecting the menu item being considered. Thus, one suggestion for speeding up learning of a new menu-driven GUI is to detect such pauses, and show (in some easily-undoable way) what the results of selecting that item would be. For instance, if choosing that item brings up a dialog box for specifying certain options, that dialog box could be shown in some grayed-out form, and would simply vanish if the user moved off that menu item. This would make the evaluation of the item much more certain, and would be an excellent guide for novice users. This is not unlike ToolTips for toolbar icons, but on a much larger scale.

A model that does an excellent job of highlighting the power of cognitive architectures is NTD-Soar (Nelson, Lehman, & John, 1994). NTD stands for "NASA Test Director", who

> ...is responsible for coordinating many facets of the testing and preparation of the Space Shuttle before it is launched. He must complete a checklist of launch procedures that, in its current form, consists of 3000 pages of looseleaf manuals...as well as graphical timetables describing the critical timing of particular launch events. To accomplish this, the NTD talks extensively with other members of launch team over a two-way radio... In addition to maintaining a good understanding of the status of the entire launch, the NTD is responsible for coordinating troubleshooting attempts by managing the communication between members of the launch team who have the necessary expertise. (p. 658)

Constructing a model that is even able to perform this task at all is a significant accomplishment. Nelson, Lehman, and John were able to not only build such a model but this

model was able to produce a timeline of behavior which closely matched the timeline produced by the actual NTD being modeled. That is, the ultimate result was a quantitative model of human performance, and an accurate one at that.

It is unlikely that such an effort could have been accomplished without the use of an integrated cognitive architecture. This was a Soar model which made use of other Soar models. Nelson, Lehman, and John did not have to generate and implement theory of natural language understanding to model the communication between the NTD and others, or the NTD reading the pages in the checklist, because one had already been constructed in Soar (Lewis, 1993). They did not have to construct a model of visual attention to manage the scanning and visual search of those 3000 pages, because such a model already existed in Soar (Weismeyer, 1992). There was still a great deal of knowledge engineering that had to go on to understand and model this complex task, but using an integrated architecture greatly eased the task of the modelers.

While this modeling effort was not aimed at a particular HCI problem, it is not difficult to see how it would be applicable to one. If one wanted to replace the 3000-page checklist with something like a personal digital assistant (PDA), how could the PDA be evaluated? There are very few NTD's in the world, and it is unlikely that they would be able to devote much time to participatory design or usability testing. However, because an appropriate quantitative model of the NTD exists, it should be possible to give the model a simulated PDA and assess the impact of that change on the model's performance. Even if the model does not perfectly capture the effects of the change, it is likely that the model would identify problem areas and at least guide the developers in using any time they have with an actual NTD.

Soar has also been used as the basis for simulated agents in wargames (Jones, et al. 1999). This model (TacAir-Soar) participates in a virtual battlespace in which humans also participate. TacAir-Soar models take on a variety of roles in this environment, from fighter pilots to helicopter crews to refueling planes. Because they are based on a cognitive architecture, they function well as agents in this environment. Their interactions are more complex than simple scripted agents, and they can interact with humans in the environment with English natural language. This is an

ambitious model containing over 5000 production rules. One of the major goals of the project is to make sure that TacAir-Soar produces human-like behavior because this is critical to their role, which is to serve as part of training scenarios for human soldiers. In large-scale simulations with many entities, it is much cheaper to use computer agents than to have humans fill every role in the simulation. While agents (other than the ubiquitous and generally-disliked paper clip) have not widely penetrated the common desktop interface, this remains an active HCI research area, and future agents in other roles could also be based on cognitive architecture rather than more engineering-oriented AI models.

There have been many other applications of Soar to HCI-related problems. Ritter, Baxter, Jones, and Young (2000) report on a number of Soar models of GUI-based tasks. So the applicability of Soar to the development of a quantitative theory of HCI is clear. Soar has particular strengths in this regard relative to the other architectures covered in this chapter. In particular, Soar is more focussed on learning than any of the other systems, and learnability is clearly an important property for many user interfaces. Second, it is known that Soar models scale up to very complex tasks, such as NTD-Soar and TacAir-Soar; tasks of this complexity have not been modeled in other architectures.

3.3 EPIC

With the possible exception of the NTD model, all of the models discussed up to this point have been almost purely cognitive models. That is, the perception and action parts of the models have been handled in an indirect, abstract way. These models focus on the cognition involved, which is not surprising given the generally cognitive background of these systems. However, even the original formulation of the MHP included processors for perception and motor control. And, in fact, user interfaces have also moved from having almost exclusively cognitive demands (e.g., one had to remember or problem-solve to generate command names) to relying much more heavily on perceptual-motor capabilities. This is one of the hallmarks of the GUI, the shift to visual processing and direct manipulation rather than reliance on complex composition of commands.

However, providing accurate quantitative models for this kind of activity requires a system with detailed models of human perceptual and motor capabilities. This is one of the major foci and contributions of EPIC (for executive process interactive control). EPIC is the brainchild of Kieras and Meyer (see especially 1996, 1997; Kieras, Wood, & Meyer, 1997). The overall structure of the processors and memories in EPIC is shown in Figure 2. This certainly bears some surface similarity to the MHP, but EPIC is substantially more detailed. EPIC was explicitly designed to pair high-fidelity models of perception and motor mechanisms with a production system. The perceptual-motor processors represent a new synthesis of the human performance literature while the production system is the same one used in the CCT work discussed earlier.

---------- insert Figure 2 about here ----------

Constructing a model in EPIC thus requires specification of both the knowledge needed in the form of production rules as well as some relevant perceptual-motor parameters. Because there are a number of processors, there are quite a number of (mostly numeric) parameters in EPIC. There are two types of parameters in EPIC: standard, which are system parameters believed to be fixed across all tasks, and typical, which are free to vary across task situations, but have more-or-less conventional values. A standard parameter in EPIC is the duration of a production cycle in the Cognitive Processor, this is 50 ms. An example of a typical value is the time it takes the Visual Processor to recognize that a particular shape represents a right arrow, which is 250 ms.

All the processors in EPIC run in parallel with one another. So, while the Visual Processor is recognizing an object on the screen, the Cognitive Processor can be deciding what word should be spoken in response to some other input, while at the same time the Manual Motor processor is pressing a key. The information flow is typical of traditional psychological models, with information coming in through the eyes and ears, and outputs coming from the mouth and hands. More specifically, what is modeled in each of the processors is primarily time course. EPIC's Visual Processor does not take raw pixels as input and compute that those pixels represent a letter "A," instead, it determines whether the object on the screen can be seen and at what level of detail, and how long it will take for a representation of that object to be delivered to EPIC's declarative

memory once the letter becomes available to the Visual Processor. The appearance of the letter can actually cause a number of different elements to be deposited into EPIC's declarative memory at different times, for example, information about the letter's color will be delivered before information about the letter's identity.

Similarly, on the motor side, EPIC does not simulate the computation of the torques or forces needed to produce a particular hand movement. Instead, what EPIC computes is the time it will take for a particular motor output to be produced after the Cognitive Processor has requested it. This is complicated by the fact that movements happen in phases. Most importantly, each movement includes a preparation phase and an execution phase. The time to prepare a movement is dependent on the number of movement features that must be prepared for each movement and the features of the last movement prepared. Features that have been prepared for the previous movement can sometimes be re-used, saving time. EPIC can make repeated identical movements rapidly because there is no feature preparation time necessary if the movements are identical. If they are not identical, the amount of savings is a function of how different the current movement is from the previous one. After being prepared, a movement is executed. The execution time for a movement corresponds roughly to the time it physically takes to execute the movement; the execution time for aimed movements of the hands or fingers are governed by Fitts's Law, which was described in section 2.1. EPIC's motor processors can only prepare one movement at a time, and can only execute one movement at a time, but may be preparing one movement while executing another. Thus, in some tasks it may be possible to pipeline movements effectively in order to generate very rapid sequences of movements.

EPIC's Cognitive Processor is a production system, the same one that was used for the earlier CCT work. One highly salient feature of this system is that multiple rules are allowed to fire on a production cycle. In fact, there is no upper bound on the number of productions that can fire on a cycle. Productions in this system are at a much higher grain size than productions in Soar, which gives EPIC a highly parallel quality at all levels. That is, all the processors work in parallel and EPIC's cognitive processor is itself capable of parallel processing.

This allows EPIC particular leverage in multiple-task situations. When more than one task is being performed, the tasks can execute in parallel. However, many of the perceptual-motor processors are effectively serial. People only have one set of eyes that can only be aimed at one place at a time, so if multiple tasks are ongoing and they both require the eyes, there must be something that arbitrates. In EPIC, this additional knowledge about how to manage multiple tasks is termed "executive" knowledge, and the productions which implement this knowledge execute in parallel with the productions implementing the task knowledge.

Why is all this machinery and extra knowledge necessary? Because the world of HCI is changing. The GUI forced designers and analysts to consider more seriously the perceptual-motor constraints, and the propagation of computers with user interfaces away from the desktop and into mobile phones, kiosks, automobiles, and many, many other places creates a huge demand on people's ability to multi-task. Multiple-task issues have largely gone unmodeled and outside the theoretical scope of most psychological accounts in HCI, at least before EPIC.

While LICAI and Soar have not been adequately equipped to deal with high-performance perception and action components of many tasks, EPIC is not equipped to handle some of the issues covered by other architectures. In particular, EPIC does not include any learning mechanisms, so it would be difficult to generate EPIC models for many of the domains Soar has approached successfully. However, this is not a fatal shortcoming, as there are a wide variety of domains in which learning is not an enormously key component and where high-fidelity modeling of perception and action, along with multiple-tasking, are central.

These are the kinds of domains to which EPIC has been applied. One of the first major applications of EPIC was to a deceptively simple dual-task paradigm known as the psychological refractory period (or PRP; see Meyer & Kieras, 1997). In this task, laboratory subjects are typically confronted with two choice reaction time tasks, something on the order of "either a red light or a green light will appear, if it's red, hit the 'L' key, if it's green, hit the 'J' key." This sounds simple, but the empirical literature is rich and shows a variety of subtle effects, for which EPIC provides the first unified account. Critically, what the EPIC models of these experiments show is that people's

low-level strategies for scheduling the tasks play a large role in determining performance in this simple paradigm.

EPIC has also been used to model some of the classic psychological results from the short-term memory (or working memory) literature (Kieras, Meyer, Muller, & Seymour, 1999). This concerns the question of how much people can remember when they repeat a string of words (or numbers) to themselves. For example, when someone reads off a telephone number to someone else who has to dial that number a few moments later, the person who has to remember the number often speaks the number repeatedly. This is called "articulatory rehearsal" in the psychology literature, and while the phenomenon had been described in detail over the years, EPIC was the first serious quantitative model of this process. An accurate model of this process is clearly important in HCI applications, as many interfaces force people to remember many things over short periods and rehearsal is a likely response to that demand. The field should have a better answer to understanding this problem than simply an admonition to "reduce working memory demand." While such an admonition is certainly good advice, more precise performance prediction is often warranted.

EPIC has been used to model several tasks with a more HCI-oriented flavor. One of those tasks is menu selection (Kieras & Meyer, 1997; Hornof & Kieras, 1997, 1999), but for brevity a detailed description of these models will be omitted. Another application of EPIC that definitely merits mention is the model of telephone assistance operators (TAOs), data originally presented in Gray, John, and Atwood (1993). When a telephone customer dials "0," a TAO is the person who answers. The TAOs modeled here sat at a "dumb terminal" style workstation and assisted customers in completing telephone calls. In particular, TAOs determine how calls should be billed, and this is done by speaking to the customer. The detailed EPIC models (Kieras, Wood, & Meyer, 1997) covered a subset of the possible billing types.

This provided a good test of EPIC because the task is performed under time pressure, and seconds—actually, milliseconds—counted in task performance. Second, this task is multimodal. The TAO must speak, type, listen, and look at a display. Third, very fine-grained performance data

were available to help guide model construction. By now it should come as no surprise to the reader that it was possible to construct an EPIC model that did an excellent job of modeling the time course of the TAO's interaction with the customer. However, this modeling effort went beyond just that and provided some insight into the knowledge engineering problem facing modelers utilizing cognitive architectures as well.

Like other production system models, EPIC provides a certain amount of freedom to the modeler in model construction. While the architecture used provides certain kinds of constraints, and these constraints are critical in doing good science and affecting the final form of the model (Howes & Young, 1997), the modeler does have some leeway in writing the production rules in the model. This is true even when the production rule model is derived from another structured representation such as a GOMS model, which was the case in the TAO model. In EPIC, it is possible to write a set of "aggressive" productions which maximize the system's ability to process things in parallel, while it is also possible to write any number of less aggressive sets representing more conservative strategies. EPIC will produce a quantitative performance prediction regardless of the strategy, but which kind of strategy should the modeler choose? There is generally no a priori basis for such a decision and it is not clear that people can accurately self-report on such low-level decisions.

Kieras, Wood, and Meyer (1997) generated an elegant approach to this problem, later termed "bracketing" (Kieras & Meyer, 2000). The idea is this: construct two models, one of which is the maximally aggressive version. At this end of the strategy spectrum, the models contain very little in the way of cognition. The Cognitive Processor does virtually no deliberation and spends most of its cycles simply reading off perceptual inputs and immediately generating the appropriate motor output. This represents the "super-expert" whose performance is limited almost entirely by the rate of information flow through the peripherals. At the other end of the spectrum, a model incorporating the slowest-reasonable strategy is produced. The slowest-reasonable strategy is one where the basic task requirements are met, but with no strategic effort made to optimize scheduling to produce rapid performance. The idea is that observed performance should fall somewhere in

between these two extremes. Different users will tend to perform at different ends of this range for different tasks, so this is an excellent way to accommodate some of the individual differences that are always observed in real users.

What was discovered by employing this bracketing procedure to the TAO models was surprising. Despite the fact that the TAOs were under considerable time pressure and were extremely well-practiced experts, their performance rarely approached the fastest possible model. In fact, their performance most closely matched a version of the model termed the "hierarchical motor-parallel model." In this version of the model, eye, hand and vocal movements are executed in parallel with one another when possible, and furthermore, the motor processor is used somewhat aggressively, preparing the next movement while the current movement was in progress. The primary place where EPIC could be faster but the data indicated the TAOs were not was in the description of the task knowledge. It is possible to represent the knowledge for this task as one single, flat GOMS method with no use of subgoals. On the other hand, the EPIC productions could represent the full subgoal structure or a more traditional GOMS model. Retaining the hierarchical representation—thus incurring time costs for goal management— provided the best fit to the TAOs performance. This provides solid evidence for the psychological reality of the hierarchical control structure inherent in GOMS analysis, since even well-practised experts in fairly regular domains do not abandon it for some kind of faster knowledge structure.

The final EPIC-only model that will be considered is the model of the task first presented in Ballas, Heitmeyer, and Perez (1992). Again, this model first appeared in Kieras and Meyer (1997), but a richer version of the model is described in more detail later, in Kieras, Meyer, and Ballas (2001). The display used is a split screen, on the right half of the display, the user is confronted with a manual tracking task which is performed using a joystick. The left half of the display is a tactical task in which the user must classify objects as hostile or neutral based on their behavior. There were two versions of the interface to the tactical task, one a command-line style interface using a keypad and one a direct-manipulation-style interface using a touchscreen. The

performance measure of interest in this task is the time taken to respond to events (such as the appearance of a new object or a change in state of an object) on the tactical display.

This is again a task well-suited to EPIC because the perceptual-motor demands are fairly extensive. This is not, however, what makes this task so interesting. What is most interesting is the human performance data: in some cases the keypad interface was faster than the touchscreen interface, and in many cases the two yielded almost identical performance, and in some other cases the touchscreen was faster. Thus, general claims about the superiority of GUIs do not apply to this case, a more precise and detailed account is necessary.

EPIC provides just the tools necessary to do this. Two models were constructed for each interface, again using the bracketing approach. The results were revealing. In fact, the fastest-possible models showed no performance advantage for either interface. The apparent direct-manipulation advantage of the touchscreen for initial target selection was almost perfectly offset by some type-ahead advantages for the keypad. The reason for the inconsistent results is that the users generally did not operate at the speed of the fastest-possible model; they tended to work somewhere in between the brackets for both interfaces. However, they tended to work more toward the upper (slowest-reasonable) bracket for the touchscreen interface. This suggests an advantage for the keypad interface, but the caveat is that the slowest-reasonable performance bound for the touchscreen was faster than the slowest-possible for the keypad. Thus, any strategy changes made by users in the course of doing the task, perhaps as a dynamic response to changes in workload, could affect which interface would be superior at any particular point in the task. Thus, results about which interface is faster are likely to be inconsistent—exactly what was found.

This kind of analysis would be impossible to conduct without a clear quantitative human performance model. Constructing and applying such a model also suggested an alternative interface which would almost certainly be faster than either, which is one using a speech-driven interface. One of the major performance bottlenecks in the task was the hands, and so voice-based interaction should, in this case, be faster. Again, this could only be clearly identified with the kind of precise quantitative modeling enabled by something like the EPIC architecture.

Despite this, the EPIC architecture is sometimes considered a less complete architecture than others because it does not include a learning mechanism, and thus would be unsuitable for the kinds of interfaces and tasks modeled with LICAI or Soar. However, LICAI and Soar might conversely be considered incomplete for their lack of detailed specification of perceptual-motor processors; the definition of a "complete" cognitive architecture is not entirely clear at this point. However, this incompleteness has been acknowledged by both the EPIC community and the Soar community, and some experimentation has been done with a union of the two architectures, using Soar in place of EPIC's Cognitive Processor. This fusion is called EPIC-Soar and some of the initial results have been promising. For example, Chong and Laird (1997) demonstrated that Soar could indeed learn at least some of the complex executive knowledge needed to manage a dual-task situation (again, a combination of a tracking task and a simple decision-making task) in EPIC. Lallament and John (1998) looked at that same task with a slightly different version of EPIC-Soar which effectively negated the cognitive parallelism found in EPIC. One of the things they found was that the cognitive parallelism in EPIC was not actually necessary for performance in this particular task; whether this is true for other tasks is still an open question. While the future of EPIC-Soar is uncertain at this point, the fact that the integration was not only possible but viable in that it has produced several running models is encouraging for a complete vision of cognitive architecture.

3.4 ACT-R/PM

ACT-R/PM (Byrne & Anderson, 1998, in press; Byrne, 2001) represents another approach to a fully unified cognitive architecture, combining a very broad model of cognition with rich perceptual-motor capabilities. ACT-R/PM is an extension of the ACT-R cognitive architecture (Anderson, 1993; Anderson & Lebiere, 1998) with a set of perceptual-motor modules like those found in EPIC (hence ACT-R/PM). ACT-R has a long history within cognitive psychology, as various versions of the theory have been developed over the years. In general, the ACT family of theories have been concerned with modeling the results of psychology experiments, and this is certainly true of the current incarnation, ACT-R. Anderson and Lebiere (1998) shows some of this

range, covering areas as diverse as list memory (chapter 7), choice (chapter 8), and scientific reasoning (chapter 11).

However, ACT-R was not originally designed to model things like multimodal, multiple-task situations like those EPIC was designed to handle. While ACT-R was certainly moving toward application to GUI-style interactions (Anderson, Matessa, & Lebiere, 1997), it was not as fully-developed as EPIC. In fact, the standard version of ACT-R is incapable of showing any kind of time-savings in even a simple dual-task situation because its operation is entirely serial. This issue is corrected with the inclusion of the PM portion of ACT-R/PM. In many ways, ACT-R/PM is a fusion of EPIC and ACT-R much in the spirit of EPIC-Soar. There are some differences, however, and the extent to which those differences result in serious differences in models in the HCI domain is not yet clear.

The ACT-R system at the heart of ACT-R/PM is, like EPIC and Soar, a production system with activity centered around the production cycle, which is also set at 50 ms in duration. However, there are many differences between ACT-R and the other architectures. First, ACT-R can only fire one production rule per cycle. When multiple production rules match on a cycle, an arbitration procedure called conflict resolution comes into play. Second, ACT-R has a well-developed theory of declarative memory. Unlike EPIC and Soar, declarative memory elements in ACT-R are not simply symbols. Each declarative element in ACT-R also has associated with it an activation value, which determines whether and how rapidly it may be accessed. Third, ACT-R contains learning mechanisms, but is not a pervasive learning system in the same sense as Soar. These mechanisms are based on a "rational analysis" (Anderson, 1990) of the information needs of an adaptive cognitive system.

For example, consider conflict resolution. Each production in ACT-R has associated with it several numeric parameters, including numbers which represent the probability that if the production fires, the goal will be reached and the cost, in time, that will be incurred if the production fires. These values are combined according to a formula that trades off probability of success vs. cost and produces an "expected gain" for each production. The matching production

with the highest expected gain is the one that gets to fire when conflict resolution is invoked. The expected gain values are noisy, so the system's behavior is somewhat stochastic, and the probability and cost values are learned over time so that ACT-R can adapt to changing environments.

Similarly, the activation of elements in declarative memory is based on a Bayesian analysis of the probability that a declarative memory element will be needed at a particular time. This is a function of the general utility of that element, reflected in what is termed its "base-level" activation, and that element's association with the current context. The more frequently and recently an element has been accessed, the higher its base-level activation will be, and thus the easier it is to retrieve. This value changes over time according to the frequency and recency of use, thus this value is learned. Associations between elements may also be learned, so that the activation an element receives based on its association with the current context can change as well. These mechanisms have helped enable ACT-R to successfully model a wide range of cognitive phenomena.

In ACT-R/PM, the basic ACT-R production system is augmented with four EPIC-like peripheral modules, as depicted in Figure 3. Like EPIC, all of these modules run in parallel with one another, giving ACT-R the ability to overlap processing. The peripheral modules come from a variety of sources. ACT-R/PM's Vision Module is based on the ACT-R Visual Interface described in Anderson, Matessa, and Lebiere (1997). This is a feature-based attentional visual system, but does not explicitly model eye movements. Recently, the Vision Module has been extended to include an eye-movement model (Salvucci, 2001a) as well. The Motor Module is nearly identical to the Manual Motor Processor in EPIC and is based directly on the specification found in Kieras and Meyer (1996), and the Speech Module is similarly derived from EPIC. The Audition module is a hybrid of the auditory system found in EPIC and the attentional system in ACT-R/PM's Vision Module.

---------- insert Figure 3 about here ----------

One other important property of ACT-R/PM is that it is possible to have ACT-R/PM interact with the same software as the human users being modeled. There are some fairly restrictive

conditions on how the software must be developed, but if these conditions are met then both the user and the model are forced to use the same software. This reduces the number of degrees of freedom available to the modeler in that it becomes impossible to force any unpleasant modeling details into the model of the user interface, because there is no model of the user interface. More will be said about this issue in the next section.

As described in the section on EPIC, multiple-tasking is becoming increasingly critical to HCI endeavors. Thus, one of the first major modeling efforts with ACT-R/PM has been to dual-task phenomena, in fact the same kinds of simple PRP dual-tasks to which EPIC has been applied (Byrne & Anderson, in press). However, as the most recent architecture of those described here, ACT-R/PM has not yet been as widely applied to HCI tasks as have the others. However, there have been some recent models that are more directly HCI endeavors than the PRP work.

The first example comes from the dissertation work of Ehret (1999). Among other things, Ehret developed an ACT-R/PM model of a fairly simple, but subtle, experiment. In that experiment, subjects were shown a target color, and asked to click on a button that would yield that color. The buttons themselves had four types: blank, arbitrary icon, text label, and color. In the color condition, the task was simple: users just found the color that matched the target, then clicked the button. In the text label condition, the task was only slightly more difficult: users could read the labels on the buttons, and select the correct one because the description matched the color. In the arbitrary icon condition, more or less random pictures appeared on each icon (e.g. a mailbox). Users had to either memorize the picture to color mapping, which they had to discover by trial and error, or memorize the location of each color, since the buttons did not change their function over time. In the hardest condition, the blank condition, users simply had to memorize the mapping between button location and color, which they had to discover through trial and error.

Clearly, the conditions will produce different average response times, and what Ehret found is that they also produced somewhat different learning curves over time. Ehret added an additional manipulation as well: after performing the task for some time, all the labeling was removed. Not surprisingly, the amount of disruption was different in the different conditions,

reflecting the amount of incidental location learning that went on as subjects performed the task. The ACT-R/PM model that Ehret constructed did an excellent job of explaining the results. This model represented the screen with the built-in visual mechanisms from ACT-R/PM and learned the mappings between color and location via ACT-R's standard associative learning mechanisms. The initial difference between the various conditions was reproduced, as were the four learning curves. The model also suffered disruptions similar to those suffered by the human users when the labels were removed. This model is an excellent demonstration of the power of ACT-R/PM, exercising both the perceptual-motor capabilities of the system as well as the graded learning in ACT-R's rational-analysis driven mechanisms.

Salvucci (2001b) describes an ACT-R/PM model that tests ACT-R/PM's ability to handle multimodal, high-performance situations in a very compelling task: this model drives an automobile driving simulator. This is not a robotics project; the ACT-R/PM model does not actually turn the steering wheel or manipulate the pedals, but rather it communicates with the automobile simulation software. The model's primary job is to maintain lane position as the car drives down the road. Salvucci (2001b) adds an additional task which makes it particularly interesting: the model dials telephone numbers on a variety of mobile phone interfaces. There were two factors, which were crossed: whether the telephone was dialed manually via keypad vs. dialed by voice, and whether the full telephone number needed to be dialed vs. a shortened "speed dial" system. The model was also validated by comparison with data from human users.

What both the model and the human users showed is that dialing while not driving is faster than dialing while driving, and that steering performance can be disrupted by telephone dialing. Not surprisingly, the most disruptive interface was the "full-manual" interface, where the full phone numbers were dialed on a keypad. This is due largely to the fact that dialing with the keypad requires visual guidance, causing the model (and the users) to take their eyes off the road. There was very little disruption associated with the voice interfaces, regardless of whether full numbers or speed-dial was used.

This is a nice illustration of the value of cognitive architectures for a number of reasons. First, the basic driving model could simply be re-used for this task; it did not have to be re-implemented. Second, the model provides an excellent quantitative fit to the human data. Third, this is an excellent example of a situation where testing human users can be difficult. Testing human drivers with interfaces that degrade driving performance is dangerous, so simulators are generally used for this kind of evaluation. However, maintaining a driving simulator requires a great deal of space and is quite expensive. If someone wanted to test another variant of the telephone interface, it would be much faster and cheaper to give that interface to Salvucci's model than it would be to recruit and test human drivers.

There is other published and ongoing work applying ACT-R/PM to HCI problems (e.g. Byrne, 2001; Schoelles & Gray, 2000), but space considerations prohibit a more exhaustive review. The vitality of the research effort suggests that ACT-R/PM's combination of perceptual-motor modules and a strong theory of cognition will pay dividends as an HCI research tool.

In fact, this is not limited to ACT-R/PM; overall, cognitive architectures are an exciting and active area of HCI research. The four systems described here all take slightly different approaches and focus on slightly different aspects of various HCI problems, but there is clearly a great deal of cross-pollination. Lessons learned and advancements made in one architecture often affect other systems, for example, the development of EPIC's peripheral systems clearly impacted both Soar and ACT-R.

3.5 Comparisons

An exhaustive and detailed comparison of the major cognitive architectures is beyond the scope of this chapter, however, an excellent comparison which includes a number of other architectures can be found in Pew and Mavor (1998). Certainly, the three production systems, Soar, EPIC, and ACT-R/PM are related.A major difference between them is their original focus; they were originally developed to model slightly different aspects of human cognition. However, as they develop, there appears to be more convergence than divergence. This is generally taken to be a good

sign that the science is cumulating.Still, there are differences, and certainly between the production systems and LICAI/CoLiDeS. Many of the relevant comparisons are summarized in Table 1.

---------- insert Table 1 about here ----------

This table does not include the hybrid EPIC-Soar system because it is not clear what is in store for that system; however, this system would essentially contain most of the attributes of Soar but would get a “yes” on the “detailed perceptual-motor systems” feature. Most of the entries on this table have been discussed previously with the exception of the last two. Support for learning will be discussed in the next section. The presence and size of the user community has not been discussed, as it is not clear what role (if any) such a community plays in the veridicality of the predictions made by the system. However, it may be relevant to researchers for other reasons, particularly those trying to learn the system.

In addition, many of the values on this table are likely to change in the future. For example, a more pervasive learning mechanism for ACT-R is slated to be a part of the next revision of the theory. Whether this will result in ACT-R being as successful as Soar in modeling things like exploratory learning is not yet clear.

It is difficult to classify the value an architecture has on a particular attribute as an advantage or a disadvantage, because what constitutes an advantage for modeling one phenomenon may be a disadvantage for modeling others. For example, consider learning in Soar. Certainly, when attempting to model the improvement of users over time with a particular interface, Soar’s learning mechanism is critical. However, there are many applications for which modeling learning is not critical and Soar’s pervasive learning feature occasionally causes undesired side effects which can make model construction more difficult.

4. The Future of Cognitive Architectures in HCI

Beyond the incremental development and application of architectures like Soar and ACT-R/PM, what will the future hold for cognitive architectures in HCI? What are the challenges faced, and what is the ultimate promise? Currently, there are indeed a number of important limitations for cognitive architectures. There are questions they cannot yet address, and questions it is hard to see

how they even would address. Other limitations are more pragmatic than in principle, but these are relevant as well.

First, there are a wide array of HCI problems that are simply outside the scope of current cognitive architectures. Right now, these architectures focus on cognition and performance, but there are other aspects of HCI, such as user preference, boredom, aesthetics, fun, and so on. Another important challenge, though one that might be overcome, is that these architectures have generally not been applied to social situations, such as those encountered in groupware or online communities (Olson & Olson, this volume; Preece, this volume). It is not in principle impossible to implement a model of social interaction in a cognitive architecture; however, the knowledge engineering problem here would certainly be a difficult one. How does one characterize and implement knowledge about social situations with enough precision to be implemented in a production system? It may ultimately be possible to do so, but it is unlikely that this will happen anytime soon.

One problem that will never entirely be resolved, no matter how diligent the modelers are, is the knowledge engineering problem. Every model constructed using a cognitive architecture still needs knowledge about how to user the interface and what the tasks are. By integrating across models, the knowledge engineering demands when entering a new domain may be reduced (the NTD is a nice example), but they will never be eliminated. This requirement will persist even if an architecture were to contain a perfect a theory of human learning—and there is still considerable work to be done to meet that goal.

Another barrier to the more widespread use of cognitive architectures in HCI is that the architectures themselves are large and complex pieces of software, and (ironically) little work has been done to make them usable or approachable for novices. For example: “EPIC is not packaged in a ‘user-friendly’ manner; full-fledged Lisp programming expertise is required to use the simulation package, and there is no introductory tutorial or user’s manual.” (Kieras & Meyer, 1997, p. 399). The situation is slightly better for Soar, which does have a frequently-asked questions Web resource (http://ritter.ist.psu.edu/soar-faq/) and some tutorial materials (http://

www.psychology.nottingham.ac.uk/staff/Frank.Ritter/pst/pst-tutorial.html). However, Soar has a reputation as being difficult to learn and use. Tutorial materials, documentation, and examples for ACT-R are available, and most years there is a two-week “summer school” for those interested in learning ACT-R (see http://act.psy.cmu.edu/). However, the resources for ACT-R/PM are somewhat more limited, though there are some rudimentary examples and documentation (see http://chil.rice.edu/byrne/RPM/).

Another limiting factor is implementation. In order for a cognitive architecture to accurately model interaction with an interface, it must be able to communicate with that interface. Because most user interfaces are “closed” pieces software with no built-in support for supplying a cognitive model with the information it needs for perception (i.e., what is on the screen where) or accepting input from a model, this creates a technical problem. Somehow, the interface and the model must be connected. A excellent summary of this problem can be found in Ritter, et al. (2000). A number of different approaches have been taken. In general, the EPIC solution to this problem has been to re-implement the interface to be modeled in Lisp, so the model and the interface can communicate via direction function calls. The ACT-R/PM solution is not entirely dissimilar. In general, ACT-R/PM has only been applied to relatively new experiments or interfaces which were initially implemented in Lisp, and thus ACT-R/PM and the interface can communicate via function calls. In order to facilitate the construction of models and reduce the modeler’s degrees of freedom in implementing a custom interface strictly for use by a model, ACT-R/PM does provide some abilities to automatically manage this communication when the interface is built with the native GUI builder for Macintosh Common Lisp under MacOS and Allegro Common Lisp under Windows. If the interface is implemented this way, both human users and the models can interact with the same interface.

Despite these limitations, this is a particularly exciting time to be involved in research on cognitive architectures in HCI. There is a good synergy between the two areas, as cognitive architectures are certainly useful to HCI, so HCI is also useful for cognitive architectures. HCI is a complex and rich yet still tractable domain, which makes it an ideal candidate for testing cognitive

architectures. HCI tasks are more realistic and require more integrated capabilities than typical cognitive psychology laboratory experiments, and thus cognitive architectures are the best theoretical tools available from psychology. Theories like EPIC-Soar and ACT-R/PM are the first complete psychological models that go from perception to cognition to action in detail. This is a significant advance and holds a great deal of promise.

Even some of the limitations have generated solid research in an attempt to overcome them. For example, Ritter, et al. (2000) describes the implementation of a generic "sim-eye" and "sim-hand" as part of a more general program researching of what they term "cognitive model interface management systems" (CMIMS). For instance, they have implemented a virtual hand and eye in Tcl/Tk which can interact with either Soar or ACT-R via a socket-based interface. This is not a trivial technical accomplishment, and may serve to help make it easier to develop computational cognitive models which interact with software systems.

The most intriguing development along this line, however, is recent work by St. Amant and Riedl (2001). They have implemented a system called VisMap which directly parses the screen bitmap on Windows systems. That is, given a Windows display—any Windows display—VisMap can parse it and identify things like text, scroll bars, GUI widgets, and the like. It also has facilities for simulating mouse and keyboard events. This is an intriguing development, because it should in principle be possible to connect this to an architecture like EPIC or ACT-R, which would enable the architecture to potentially work with any Windows application in its native form.

While there are still a number of technical details which would have to be worked out, this has the potential of fulfilling one of the visions held by many in the cognitive architecture community: a high-fidelity "virtual user" that could potentially use any application or even combination of applications. Besides providing a wide array of new domains to researchers, this could be of real interest to practitioners as well because this opens the door for at least some degree of automated usability testing. This idea is not a new one (e.g. Byrne, Wood, Sukaviriya, Foley, & Kieras, 1994; St. Amant, 2000), but technical and scientific issues have precluded its adoption on even a limited scale. This would not eliminate the need for human usability testing (see Ritter &

Young, 2001, for a clear discussion of this point) for some of the reasons listed above, but it could significantly change usability engineering practice in the long run.

The architectures themselves will continue to be updated and applied to more tasks and interfaces. There is a new version (version 5.0) of ACT-R currently under development, and this new version has definitely been influenced by issues raised by the PM system and numerous HCI concerns. New applications of EPIC result in new mechanisms (e.g., similarity-based decay in verbal working memory storage, Kieras, et al., 1999) and new movement styles (e.g. click-and-point, Hornof & Kieras, 1999). Applications like the world-wide web are likely to drive these models into more semantically rich domain areas, and tasks which involve greater amounts of problem-solving are also likely candidates for future modeling.

The need for truly quantitative engineering models will only grow as user interfaces propagate into more and more places and more and more tasks. Cognitive architectures, which already have a firmly-established niche in HCI, seem the most promising road toward such models. Thus, as the architectures expand their range of application and their fidelity to the humans they are attempting to model, this niche is likely to expand. HCI is an excellent domain for testing cognitive architectures as well, so this has been, and will continue to be, a fruitful two-way street.

References

Altmann, E. M. (2001). Near-term memory in programming: A simulation-based analysis. *International Journal of Human-Computer Studies, 54(2),* 189-210.

Altmann, E. M., & John, B. E. (1999). Episodic indexing: A model of memory for attention events. *Cognitive Science, 23(2),* 117-156.

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

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

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science, 13(4),* 467-505.

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

Anderson, J. R., Matessa, M., & Lebiere, C. (1997). ACT-R: A theory of higher level cognition and its relation to visual attention. *Human-Computer Interaction, 12(4),* 439-462.

Ballas, J. A., Heitmeyer, C. L., & Perez, M. A. (1992). Evaluating two aspects of direct manipulation in advanced cockpits. *Proceedings of ACM CHI 92 Conference on Human Factors in Computing Systems* (pp. 127-134). New York: ACM.

Bovair, S., Kieras, D. E., & Polson, P. G. (1990). The acquisition and performance of text-editing skill: A cognitive complexity analysis. *Human-Computer Interaction, 5(1),* 1-48.

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

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

Byrne, M. D., Anderson, J. R., Douglass, S., & Matessa, M. (1999). Eye tracking the visual search of click-down menus. *ACM CHI'99 Conference on Human Factors in Computing Systems* (pp. 402-409). New York: ACM.

Byrne, M. D., & Bovair, S. (1997). A working memory model of a common procedural error. *Cognitive Science, 21(1),* 31-61.

Byrne, M. D., Wood, S. D., Sukaviriya, P. N., Foley, J. D., & Kieras, D. E. (1994). Automating interface evaluation. *ACM CHI'94 Conference on Human Factors in Computing Systems* (pp. 232-237). New York: ACM Press.

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

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

Ehret, B. D. (1999). Learning where to look: The acquisition of location knowledge in display-based interaction. Unpublished doctoral dissertation, George Mason University, Fairfax, VA.

Franzke, M. (1994). Exploration, acquisition, and retention of skill with display-based systems. Unpublished doctoral dissertation, University of Colorado, Boulder.

Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world task performance. *Human-Computer Interaction, 8(3),* 237-309.

Gray, W. D., Young, R. M., & Kirschenbaum, S. S. (1997). Introduction to this special issue on cognitive architectures and human-computer interaction. *Human-Computer Interaction, 12,* 301-309.

Hornof, A., & Kieras, D. E. (1997). Cognitive modeling reveals menu search is both random and systematic. *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems* (pp. 107-114). New York: ACM.

Hornof, A., & Kieras, D. (1999). Cognitive modeling demonstrates how people use anticipated location knowledge of menu items. *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems* (pp. 410-417). New York: ACM.

Howes, A., & Young, R. M. (1996). Learning consistent, interactive, and meaningful task-action mappings: A computational model. *Cognitive Science, 20(3),* 301-356.

Howes, A., & Young, R. M. (1997). The role of cognitive architecture in modeling the user: Soar's learning mechanism. *Human-Computer Interaction, 12(4),* 311-343.

Huguenard, B. R., Lerch, F. J., Junker, B. W., Patz, R. J., & Kass, R. E. (1997). Working memory failure in phone-based interaction. *ACM Transactions on Computer-Human Interaction, 4,* 67-102.

John, B. E., & Kieras, D. E. (1996). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction, 3,* 320-351.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., & Koss, F. V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine, 20(1),* 27-41.

Just, M. A., & Carpenter, P. A. (1992). A capacity theory of comprehension: Individual differences in working memory. *Psychological Review, 99(1),* 122-149.

Kieras, D. E., & Bovair, S. (1986). The acquisition of procedures from text: A production-system analysis of transfer of training. *Journal of Memory & Language, 25(5),* 507-524.

Kieras, D. E., & Meyer, D. E. (1996). The EPIC architecture: Principles of operation. Available: ftp://ftp.eecs.umich.edu/people/kieras/EPICarch.ps.

Kieras, D. E., & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction, 12(4),* 391-438.

Kieras, D. E., & Meyer, D. E. (2000). The role of cognitive task analysis in the application of predictive models of human performance. In J. M. Schraagen & S. F. Chipman (Eds.), *Cognitive task analysis* (pp. 237-260). Mahwah, NJ: Erlbaum.

Kieras, D. E., Meyer, D. E., Mueller, S., & Seymour, T. (1999). Insights into working memory from the perspective of the EPIC architecture for modeling skilled perceptual-motor and cognitive human performance. In A. Miyake & P. Shah (Eds.), *Models of working memory: Mechanisms of active maintenance and executive control* (pp. 183-223). New York: Cambridge University Press.

Kieras, D. E., Meyer, D. E., & Ballas, J. A. (2001). Towards demystification of direct manipulation: cognitive modeling charts the gulf of execution, *Proceedings of ACM CHI 01 Conference on Human Factors in Computing Systems* (pp. 128-135). New York: ACM.

Kieras, D., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22(4),* 365-394.

Kieras, D. E., Wood, S. D., & Meyer, D. E. (1997). Predictive engineering models based on the EPIC architecture for multimodal high-performance human-computer interaction task. *Transactions on Computer-Human Interaction, 4(3),* 230-275.

Kintsch, W. (1998). *Comprehension: A paradigm for cognition.* New York: Cambridge University Press.

Kitajima, M., & Polson, P. G. (1997). A comprehension-based model of exploration. *Human-Computer Interaction, 12(4),* 345-389.

Kitajima, M., Blackmon, M. H., & Polson, P. G. (2000). A comprehension-based model of web navigation and its application to web usability analysis. In S. McDonald & Y. Waern & G. Cockton (Eds.), *People and Computers XIV - Usability or Else!* (Proceedings of HCI 2000) (pp. 357-373). New York: Springer.

Lallement, Y., & John, B. E. (1998). Cognitive architecture and modeling idiom: An examination of three models of the Wickens's 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.

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review, 104(2),* 211-240.

Lewis, R. L. (1993). An architecturally-based theory of human sentence comprehension. Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

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

Nelson, G., Lehman, J. F., & John, B. E. (1994). Integrating cognitive capabilities in a real-time task. In A. Ram & K. Eiselt (Eds.), *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society* (pp. 353-358). Hillsdale, NJ: Erlbaum.

Newell, A. (1973). You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In W. G. Chase (Ed.), *Visual information processing.* New York: Academic Press.

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

Newell, A., & Simon, H. A. (1963). GPS, a program that simulates human thought. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought* (pp. 279-293). Cambridge, MA: MIT Press.

Pew, R. W., & Mavor, A. S. (Eds.). (1998). *Modeling human and organizational behavior: Application to military simulations.* Washington, DC: National Academy Press.

Pirolli, P., & Card, S. (1999). Information foraging. *Psychological Review, 106(4),* 643-675.

Polk, T. A., & Newell, A. (1995). Deduction as verbal reasoning. *Psychological Review, 102(3),* 533-566.

Polson, P. G., Lewis, C., Rieman, J., & Wharton, C. (1992). Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies, 36(5),* 741-773.

Polson, P. G., Muncher, E., & Engelbeck, G. (1986). A test of a common elements theory of transfer. *Proceedings of ACM CHI'86 Conference on Human Factors in Computing Systems* (pp. 78-83). New York: ACM.

Rieman, J., Young, R. M., & Howes, A. (1996). A dual-space model of iteratively deepening exploratory learning. *International Journal of Human-Computer Studies, 44(6),* 743-775.

Ritter, F. E., Baxter, G. D., Jones, G., & Young, R. M. (2000). Cognitive models as users. *ACM Transactions on Computer-Human Interaction, 7,* 141-173.

Ritter, F. E., & Young, R. M. (2001). Embodied models as simulated users: Introduction to this special issue on using cognitive models to improve interface design. *International Journal of Human-Computer Studies, 55,* 1-14.

Salthouse, T. A. (1988). Initiating the formalization of theories of cognitive aging. *Psychology & Aging, 3(1),* 3-16.

Salvucci, D. D. (2001a). An integrated model of eye movements and visual encoding. *Cognitive Systems Research, 1(4),* 201-220.

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

Schoelles, M. J., & Gray, W. D. (2000). Argus Prime: Modeling emergent microstrategies in a complex simulated task environment. In N. Taatgen & J. Aasman (Eds.), *Proceedings of the Third International Conference on Cognitive Modeling* (pp. 260-270). Veenendal, NL: Universal Press.

Simon, H. A. (1996). *The sciences of the artificial* (3rd ed.). Cambridge, MA: MIT Press.

Singley, M. K., & Anderson, J. R. (1989). *The transfer of cognitive skill.* Cambridge, MA: Harvard University Press.

St. Amant, R. (2000, Summer). Interface agents as surrogate users. *Intelligence* magazine, 11(2), 29-38.

St. Amant, R., & Riedl, M. O. (2001). A perception/action substrate for cognitive modeling in HCI. *International Journal of Human-Computer Studies, 55,* 15-39.

Tullis, T. S. (1983). The formatting of alphanumeric displays: A review and analysis. *Human Factors, 25(6),* 657-682.

Wiesmeyer, M. (1992). An operator-based model of covert visual attention. Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

Young, R. M., Barnard, P., Simon, T., & Whittington, J. (1989). How would your favorite user model cope with these scenarios? *ACM SIGCHI Bulletin, 20,* 51-55.

Table 1: Architecture feature comparison.

	LICAI/ CoLiDeS	Soar	EPIC	ACT-R/PM
Original focus	text comprehension	learning and problem-solving	multiple-task performance	memory and problem-solving
Basic cycle	construction-integration	decision cycle	production cycle (parallel)	production cycle (serial)
Symbolic or activation-based?	both	symbolic	symbolic	both
Architectural goal management	special cycle types, supports vague goals	universal subgoaling	none	goal stack
Detailed perceptual-motor systems	no	no	yes	yes
Learning mechanisms	no	yes, pervasive	no	yes, but not pervasive
Large, integrated models	no	yes	no	no
Extensive natural language	yes	yes	no	no
Support for learning system	none	FAQ, some tutorial materials	none	extensive tutorial materials, summer school
User community*	none	some, primarily AI	none	growing, primarily psychology

* Outside of the researchers who have developed the system.

Figure 1. Model Human Processor. Based on Card, Moran, and Newell (1983).

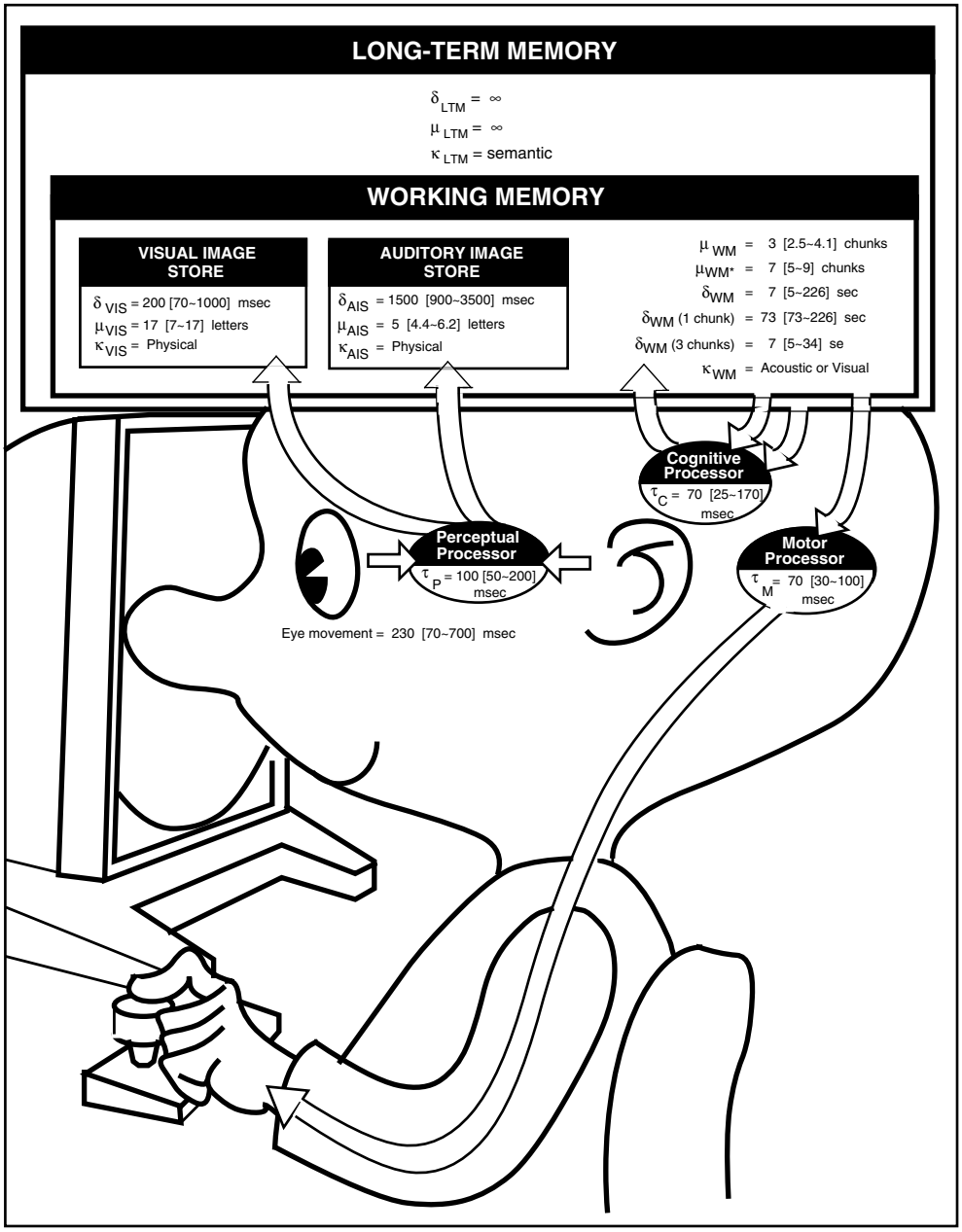

Figure 2. Overall structure of the EPIC architecture. Based on Kieras and Meyer (1996).

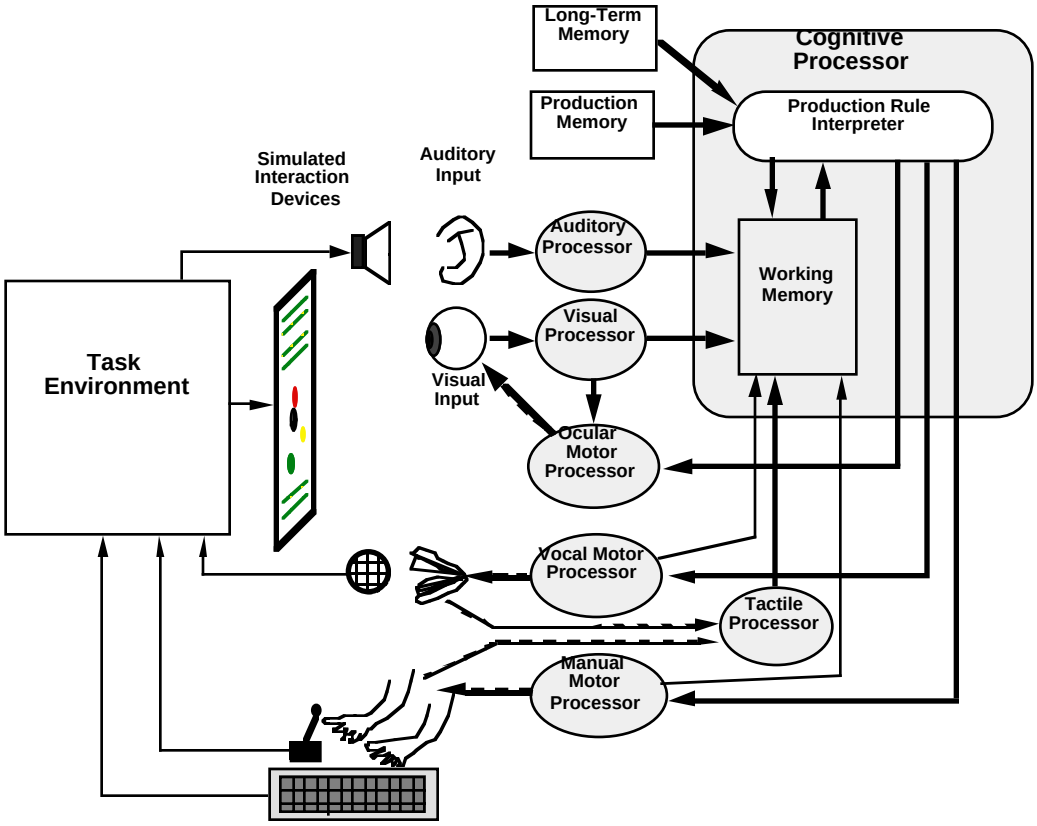

Figure 3. Overall structure of the ACT-R/PM architecture. Based on Byrne and Anderson (1998).