Running Head: THE MINIMAL CONTROL PRINCIPLE

The Minimal Control Principle

Niels Taatgen

Carnegie Mellon University and University of Groningen

Abstract

Control in cognitive models is usually fully internal, and tied to a goal representation. To explain human flexibility and robustness in task performance, however, control should be shared between the goal (top-down control) and perceptual input (bottom-up control). According to the *Minimal Control Principle*, top-down control should be minimized in order to obtain optimal flexibility with the smallest set of task knowledge. In the ACT-R architecture the amount of control can be quantified in the number of control states needed in the goal. Some examples of small tasks will be discussed, and the consequences for larger tasks and for learning.

The Minimal Control Principle

Cognitive models based on production rules have often been criticized (e.g., Dreyfus, 1979; Clark, 1997) for disregarding the need to interact with the outside world and solve problems completely “in the head”. In these models interaction with the outside world only consists of perceiving the initial state and producing a sequence of actions to reach the goal. Many studies (e.g., Larkin & Simon, 1987; Hutchins, 1995; Gray & Fu, 2004) have stressed the importance of external representations in cognition and the need for cognitive models to be able to interact and produce such external representations. In parallel to this, developments in robotics and situated cognition (Clark, 1997) have made it clear that interaction with the world is an essential element of cognition that cannot be dismissed even in models of higher cognition. In recent years we have seen a development in which many cognitive architectures have been outfitted with modules to realistically interact with the outside world (e.g., ACT-R/PM: Byrne & Anderson, 2001; EPIC-Soar: Chong & Laird, 1997) or architectures whose sole focus is on this interaction (e.g., EPIC: Meyer & Kieras, 1997). These new architectures enable models that accurately predict human performance and learning in many situations in which humans have to interact with the outside world, usually a computer screen, keyboard and mouse. Despite their success, most models only partially achieve the embodied interaction goal, because the *control* of action is still a strictly internal affair. This is not necessarily a limitation of the underlying architecture, but maybe more of current practice in cognitive modeling. The goal of this chapter is to promote a different way constructing cognitive models. I hope to convince the reader that people not only offload as much as possible of the problem representation to the environment, but also as much of control as possible, and that therefore our models should do the same to be accurate.

The example of making coffee

To illustrate the problem I would like to go through an example by Larkin (1989). Larkin describes the task of making coffee, which on the surface seems to be an easy and effortless task, but proves to be quite complex in detail. Figure 1 shows some of the subtasks of making coffee with their respective dependencies. Given the complexity of the graph, it is amazing that the process itself, assuming it is carried out by and experienced coffee maker, is almost completely effortless. Larkin lists the following properties that require an explanation:

1. The process is easy.
2. It is largely error-free.
3. It is not degraded by interruption: the task can be picked up at any stage without extensive problem solving, and it is even possible to complete the coffee-making process when someone else has started it.
4. The steps are performed in a variety of orders. The constraints on the task only partially order the subtasks, allowing several orders of executing the steps, and in theory even parallel execution of steps.
5. The process is easily modified. For example, if there is still water in the reservoir, the plan can effortlessly be modified to skip filling the reservoir but otherwise carry out all the other steps.
6. Performing the task smoothly and easily requires learning. This means that the reasoning process doesn't come "for free". On the other hand, experts can adopt the plan in novel ways, so it is not necessarily a case of caching old solutions.

The important aspect of Larkin's model DiBS (Display-based solver) is that it does not retain information that can be observed in the environment in its working memory. It is therefore very flexible in its planning and can handle interruptions very well. Larkin also observes that errors made in making coffee typically involve properties that cannot be perceived. For example, if the reservoir of the coffee maker is transparent, people never try to fill it twice, but when it is opaque, a frequent error is to try to fill a reservoir that is already full.

What Larkin's example illustrates is that in making coffee not only part of the representation of the task is offloaded to the world, but also part of the control over the task. It is not the case that our actions are solely determined by our internal goals, but also partly by the environment. The example also makes clear that some external control on our actions is highly beneficial. Another example in which external control is useful is in multi-tasking situations. People generally are quite proficient in executing multiple well-practiced tasks in parallel as long as they do not use the same perceptual and motor resources at the same time, even if those tasks have been learned separately. This can either be explained by a very sophisticated planning algorithm, or by the fact that the external presence of multiple tasks facilitates their execution.

Control and the Representation of Instructions

Let us analyze the slightly more simple example of brewing tea. An elementary cookbook might give the following recipe:

1. Put water in kettle
2. Put kettle on stove
3. Put leaves in tea pot
4. Wait until water boils
5. Pour water in tea pot

If we would know nothing about the function of all these steps (which can be the case in some more involved recipes), the only thing that we could do is carry out the instructions in their listed order, while keeping the last step we have performed in memory, which then serves as a cue for determining the next step. Such a problem representation requires six control states, a start and end state, and four states in between the five steps. A control state in this context means that we have an internally represented symbol or marker of where we are in the process. This representation leads to inflexible behavior, and would easily break down. For example, if we would lose track of the state due to some interruption, it would be hard to pick up the task again. Also, this representation hides the fact that step 3 can be done in parallel with any of steps 1, 2 and 4, allowing many different orderings that are potentially more efficient.

A better representation is to include in each step the conditions under which that step can be taken, producing the following representation for the tea example:

1. [empty kettle] put water in kettle
2. [kettle with cold water] put kettle on stove
3. [empty teapot] put leaves in teapot
4. [water boils and leaves are in teapot] pour water in teapot

The big advantage of this representation is that control is shifted to the environment, so it is no longer necessary to keep track of where you are: the environment will cue the next step. It also allows for multiple orderings of the steps, and saves out one instruction, because it is no longer necessary to specify the “wait” step. Instead of six control states, we only need a single control state. From this example, and from Larkin’s analysis of the coffee task, it seems that minimizing control states is a good idea in general. It led me to formulate (Taatgen, 2005) the *Minimal Control Principle*:

When people have to do a new task, they strive for a strategy with a minimal number of control states

The minimal control principle refers to human cognition, but is formulated in terms of a cognitive modeling construct, a control state. To support this idea consider the following analysis. This analysis assumes that we use some sort of production-based system. The set of production rules that we specify produces the following mapping:

P: {internal states} x {sensory inputs} --> {actions}

Given some internal state, and a set of sensory inputs, the set of productions rules specify the actions that should be taken. This is similar to Card, Moran and Newell's (1983) "Goals + Task + Operators + Input + Knowledge + Process-limits --> Behavior mapping", but for the moment only distinguishing internal from external conditions. A useful subdivision in the internal state of the system is to subdivide the internal state into a control state and a problem state. The problem state stores specific information about the problem (i.e., what are the numbers to be added or what is the flavor of the tea we are making), and the control state keeps track of where we are in the problem solving process. The expanded mapping is:

P: {control states} x {problem states} x {sensory inputs} --> {actions}

One way to judge the robustness of a production system is to see whether the rules in the system have appropriate actions for any conceivable combination of control state, problem state and sensory input, preferably with as few as possible individual productions. This means that number of control states, problem states and possible sensory inputs all influence the eventual complexity and robustness of the production system. Sensory input is more or less out of our control, although an appropriate attentional mechanism might be able to filter out irrelevant input. The impact of the number of possible problem states can potentially be limited, because a single production rule can handle many problem states by using variables. For example, one addition production rule can handle any combination of addends in the problem state. That leaves us with the number of control states. Because it usually much harder if not impossible to use variables in matching control states, the number of production rules needed to provide a full coverage of all possible states and inputs increases linearly with the number of control states. Effectively this means that we have to create rules for exceptions that may or may not occur in reality, and rules for picking up the task after an interruption. For example, in the case where we are in the start state of making tea, but there is still enough water in the kettle, we would need a separate rule to cover this case if we have six control states, but not if we have only one control state. It is important to keep in might that minimal control is a principle of human cognition. Under the assumption that people strive for minimal control, it should therefore also be a guiding principle for constructing models.

Although minimal control seems to be intuitively right, it is nevertheless at odds with present modeling practice. Many methods for describing the structure of tasks organize their descriptions in hierarchies of steps. GOMS, (Card, Moran & Newell, 1983) for example uses methods to organize the structure of the task, where each methods consists of a sequence of operators. Each of these operators can be either a primitive step or a call to a new method. When a GOMS-like analysis serves as the basis for a cognitive model, the result is a model in which each operator in each method needs its own control state. Card, Moran and Newell (1983, p. 147) note the problem themselves: “For a general treatment of errors and interruptions of the user, the hierarchical control structure of a GOMS model is inadequate; a more general control structure is required. The use of the stack-discipline GOMS model (...) should be taken as an approximation here because of its greater simplicity.”

Minimal Control within the ACT-R Architecture

The Minimal Control Principle is independent of the particulars of the modeling paradigm or architecture that is used. For our present purposes we will use ACT-R (Anderson et al., 2004) to go through some examples, and also discuss the issue of learning. Recent changes to the architecture (ACT-R 6) make it easier to construct models that partially derive control from external input, so it therefore particularly suitable to illustrate the points I want to make. The ACT-R architecture has a modular structure with a central production system (Figure 2). A key property of the architecture is that the central production system communicates with the other modules through buffers that can only hold single pieces of information at a time. For example the visual buffer holds a representation of what is currently visually attended, the manual buffer holds a representation of the manual action that is currently carried out, and the declarative buffer holds the most recently retrieved fact from memory. The production system can inspect all of these contents (but has no further access to the modules behind them), and determine actions that take the form of buffer modifications. In ACT-R the behavioral mapping discussed earlier is simply:

$\{buffer_1\} \times \{buffer_2\} \times \ldots \times \{buffer_n\} \text{ --> } \{buffer_1\} \times \{buffer_2\} \times \ldots \times \{buffer_n\}$

Some buffers refer to representations of sensory inputs, others to the problem state, and others to the control state. More specifically, the goal buffer is generally used to hold the control state as well as part of the problem state. However, many recent models (i.e., Anderson, 2005) now have a separate problem state buffer, because fMRI research has indicated that the control and problem state are represented in different parts of the brain. If we adopt this practice, we can make the ACT-R mapping more specific:

$\{\text{goal buffer}\} \times \{\text{problem state buffers}\} \times \{\text{perceptual/motor buffers}\} \text{ --> } \{\text{buffer modifications}\}^n$

The number of control states can now easily be specified as the number of possible values that the goal buffer can take. Let us walk through an example to clarify this. Suppose we have simple choice reaction task paradigm, in which a letter appears on the screen, and the subject has to push “D” if the letter is an “A”, and “K” when the letter is a “B”. Table 1 lists two production systems to do this task, the first row of the table a system with three control states, and the second row a production system with one state. The condition of each of the rules (before the arrow) consists of several buffer matches, each starting with =buffer> or ?buffer>. The condition part of the *perceive* rules have three conditions. The first condition is to check the goal or control state, the second whether the visual module is not busy, and the third whether a new visual element is present on the screen. If these conditions hold, the rule will direct attention to the stimulus by a buffer action (the +visual> part of the action). This rule basically waits until a stimulus appears on the screen, and will then direct attention to it. Once the visual module has moved attention, and has perceived the letter, it will place this information in the visual buffer, allowing the second rule to act. This rule matches the perceived letter, and then tries to retrieve the letter from declarative memory (note that variables start with an “=”). We assume that the letter mappings “A” to “D” and “B” to “K” are stored as facts in declarative memory. Once declarative memory has found the appropriate fact, and has placed it in the retrieval buffer, the third rule can match this, and issue a key press. The “-goal>” in the third rule clears the goal buffer, indicating that the task has been accomplished. The two production systems are virtually identical, except that the first system updates the state slot in the goal buffer after each rule. The second system relies on a feature that has been introduced in ACT-R 6, which is that most buffers (with the exception of the goal buffer) are automatically cleared after they have been read. After the “attend” rule has fired, the visual-location buffer is cleared, ensuring that the rule will not fire a second time. Another aspect of the architecture that can be illustrated by this small example is that the different modules function asynchronously. Figure 3 shows the activity of the different modules and the central rule system in the execution of this module (for the case where an “A” is displayed). A production rule firing takes 50 ms, the time to attend a visual stimulus 85 ms. The time to retrieve a fact from memory depends on its activation (see Anderson et al, 2004 for details), and the manual time depends on the current position of the hand and the size of the target. In this example execution is fully serial, which can be attributed to the task because no step can be done before the previous step has finished.

Despite the small differences between the two models, the single state model has a huge advantage: it can be interrupted, because there is no control state to reconstruct after the interruption. This property also makes it possible to perform a secondary task in parallel to this task without too much performance loss. Figure 3 shows that most of the modules have time to spare during the task, making it in principle possible to do more tasks at the same time. Suppose we have a second task, which involves responding to a tone by saying a word, where different tones map onto different words. This paradigm, where a visual-manual task and an aural-vocal task have to be done in parallel, has been used by Schumacher et al. (2001) and by Hazeltine, Teague and Ivry (2002) to show that with sufficient practice subjects can perfectly parallelize two tasks as long as there is no overlap in perceptual/motor resources. In order to handle these two tasks, our one control-state model needs three additional rules to handle an audio input and produce a vocal output, but which are otherwise identical to the three rules for a visual input and manual output. Augmenting the three-state model would be much harder. One solution is to increase the number of control states to nine for every possible combination of the three states of the individual tasks. For example, we would need a control state that represents the fact that we are retrieving a visual-manual mapping and are perceiving the tone. Two production rules for each of these states would then be needed, in the example one for the case in which the retrieval of the mapping finishes first, and one for the case in which the perception of the tone finishes first.

How Skilled Performance is Learned

In the dual-task situation, the two tasks compete for two central resources: the production system and declarative memory. Contrary to the perceptual and motor resources whose execution times are more or less fixed by the physical limitations of the system, ACT-R has learning mechanisms to reduce the load on central resources. Indeed, the production systems in Table 1 only model a certain stage in the learning, where it is assumed that the general task is represented as production rules, but the specifics of the stimulus-action mappings are stored in declarative memory. The general ACT-R assumption is that all task-specific knowledge is initially fully declarative, but in the limit fully procedural (Taatgen & Lee, 2003; Anderson et al., 2004). It is important to incorporate learning in our discussion of the minimal control principle because of Larkin's observation that smooth performance requires learning, which is of course a well-known fact of skill acquisition. The explanation for this fact is that the production rules in Table 1 are all matched in parallel, so that the rule that matches the right combination of internal state and external input is automatically selected. Such parallel matching is however not possible in declarative memory. Table 2 shows a declarative representation of the choice reaction task. For explanation purposes this representation has been kept very simple, consisting of just the task, the condition and the action. For larger models this representation has to be extended with arguments for both the condition and the action. Although this representation of instructions is similar to our earlier work (Taatgen & Lee, 2003; Anderson et al., 2004), it differs in one aspect: the instructions are unordered. But let us first look at how instruction interpretation functions. A key difference between interpreted declarative instructions and production rules is that the instruction first has to be retrieved before its condition can be checked. The first rule in Table 3 tries to retrieve an instruction for the current task from declarative memory. This retrieval request matches any instruction for the current task, but in many cases spreading activation can help finding the most appropriate instruction. Suppose this production retrieves instruction *Step1* from Table 2. This instruction has as condition that there has to be stimulus present that the visual system has noticed, or, in ACT-R terms, that the visual-location buffer has a value in it. Having retrieved this instruction, there are two possibilities: either there is a visual-location, in which case we can proceed and attend it, or there is no visual-location, meaning we have to retrieve another instruction. The former is taken care of by the *implement-visual* rule, which matches the instruction and the visual-location, and issues the move-attention command to the visual module. The latter is taken care of by the third rule, *no-visual-location*, which notices that the visual-location buffer is empty, and therefore asks declarative memory to find another instruction. The consequence is that when a model is interpreting instructions, it may have to search for the right instruction first. Another possibility, which I will discuss later, is that there is no applicable instruction, because it has been forgotten or never been given. In that case a proper next step has to be inferred or discovered. Summarizing, initial behavior is smooth nor fast, because declarative retrieval is slow, the next step sometimes has to be searched for, or worse, reconstructed, inferred or discovered.

The learning mechanism that can turn the slow and brittle novice behavior into skilled behavior is called *production compilation* (Taatgen & Anderson, 2002). Production compilation combines pairs of rules that fire in sequence into new rules that combine the conditions and actions of both rules. If the first rule involves a declarative retrieval which is then used in the condition of the second rule, this retrieval is substituted into the new rule. As a consequence, the learned rules are more specialized than the original rules, and declarative retrievals are eliminated from the process. For example, on the basis of the left two rules in Table 3, and Step1 in Table 2, the following rule is learned:

```
(p learned-rule
   =goal>
        isa goal
        task crt
   ?visual>
        state free
   =visual-location>
        isa visual-location
==>
   +visual>
        isa move-attention
        screen-pos
            =visual-location)
```

This rule combines the conditions and actions of the two original rules, except for the references to the retrieval buffer. Instead, in this case one variable from the original rules (=task) is substituted for a constant from the retrieval (crt). Apart from the advantage of saving both the retrieval time of the instruction the execution time of a production rule, this rule is applicable whenever its conditions are satisfied. This shift in representation can be quite significant in situations in which the order of actions is not fixed. In the case of the dual-tasking paradigm, a declarative representation not necessarily produces the optimal order of steps, but the procedural representation does, because each event is handled as soon as possible. Taatgen (2005) and Anderson, Taatgen & Byrne (2005) describe models that start out with declarative instructions, and learn to perfectly interleave a visual-manual and an aural-vocal task in such a way that dual-task performance equals single-task performance. Figure 4 shows an example of such a perfect integration.

Control States in more Complex Tasks

The examples that we have discussed up to here have no need for control states, because other events contain enough information to determine what to do next. In more complex cases this is often not true, making it necessary to have a number (albeit minimal) of control states. An extra control state is necessary if two situations that are identical with respect to perceptual input and problem state require different actions. Also, even if the next action can in principle be derived from the environment, it might take effort to collect this information, in which case an extra control state is more efficient (see Gray & Fu, 2004, for the case of using internal versus external representations). A control state may also be needed if we do not want a certain process to be interrupted by external events. One way to achieve this is to structure instructions in a *weak hierarchy* (Taatgen, 2005). In this representation instructions are organized in *rule sets*, where each rule set has one control state that is shared by all the instructions that are in that set. In Taatgen (2005), I describe the example of a radar-screen operator, who has to choose radar tracks on a screen, identify them and enter the classification into the system. For each of these stages the model uses a separate rule set, but within such a set a single control state is used (Figure 5). The instructions in each set are organized so that the most likely order of retrieval is the order in which they are learned (using ACT-R's spreading activation mechanism), indicated by the dotted line in the figure. The interesting aspect of this model is that as it gradually transforms declarative instructions into task-specific production rules, the model starts to exhibit new strategies. For example, part of the radar task involves clicking on the track on the screen that you want to work on. The rule set that specifies this part of the task involves two steps: visually attending an unclassified track on the screen, and clicking it with the mouse. The attention step has no conditions, while the clicking step needs an attended track. The activation settings for this rule set encourage to retrieve the attention step first, and then the clicking step. Once these steps are learned as production rules this ordering disappears because activation no longer plays a role, and interesting new behavior develops: first the model attends a track, then it starts moving the mouse towards it, but during this movement it attends a new track. A comparison process that compares the old to the new track to see which of the two is currently more important, and will switch to the newer track if that one turns out to be more important. The model exhibits various other qualitative improvements in behavior, like adopting a strategy of using the hands and keyboard and skipping scanning of parts of the display that are always the same. These improvements can be attributed to the loose control structure that allows perceptual input to guide behavior.

Let us look at the slightly different example of programming a VCR by Gray (2000, Gray & Fu, 2004). The idea behind the minimal control principle is that when people approach a new task they try to find a task representation that requires as few control states as possible. This fits in well with what Gray calls the least-effort principle of cognitive engineering. Programming a VCR involves entering four pieces of information: the channel number, the day of the recording, the start time and the end time. Although it is conceivable to design a VCR in which all of this information can be entered in a mode-less manner, allowing a task representation with a single control state, this is not possible for the particular VCR in the Gray-study. This VCR has a mode switch with three positions: start, end and clock-set. The channel, day and start-time have to be entered while the switch is in the start position. The end-time has to be entered while the switch is in the end position using the same keys and display as are used for entering the start time. When all the information has been entered, the switch has to be set to *clock-set*, after which the *prog rec* button has to be pushed. The consequence is that this design enforces a three control state representation. In the Gray experiment subjects first have to use the interface without instruction, and in that phase they often do not use the mode switch at all, despite the fact that they use all the other task-based controls. Errors that are made, even after instruction, are often control state related. For example, once the mode switch is set to end-time, it is no longer possible to change the channel, but subjects still sometimes try to do this.

Structuring a task in terms of control states shows a close resemblance to the definition of a unit task. Although in many practical applications unit tasks are often defined by modelers and designers, in the original Card, Moran and Newell (1983) definition the unit task is a construct defined from the perspective of a user (p. 140): “Although it is often possible to predict the user’s actual segmentation of the task into subtasks from the way the instructions are expressed (...), it is worth emphasizing that the definition of the subtasks is a decision of the user. We use the term *unit task* to denote these user-defined subtasks.” Card, Moran and Newell give three reasons for structuring tasks into subtasks: working memory capacity, information horizons and error control. Working memory capacity, which is related to the problem state in ACT-R, can be a reason to divide a task into subtasks, because the task as a whole needs too many temporary data elements. A control state can help to partition these elements. For example, in the VCR task it is still in principle possible to use one control state if keeping looking at the setting of the mode switch is part of every step, but it is more efficient to incorporate it in a control state. Information horizon refers to the problem that the number and length of steps to be taken have to be within reasonable limits, otherwise the number of potential steps that can be taken becomes to large, and searching for the right step takes too long. Also, in cases where the right step (we will look at this problem in the next section) still has to be constructed it is not feasible if the task is too extensive. Delimiting unit tasks finally makes it easier to localize errors, assuming the unit tasks have been chosen well. This seems to suggest that credit assignment should take place at unit tasks boundaries, i.e., during changes in control state.

Each of these reasons are pressures to increase the number of control states, while the minimal control principle is a pressure to keep the number of control states as low as possible. The Gray study suggests that people approach new tasks by starting with one control state, and then gradually add more if the need arises.

Inferring Steps in incomplete Task Representations

The discussion up to this point assumes that all the steps that have to be done to accomplish the goal are present in declarative memory. In reality this assumption is too optimistic: in many cases steps have to be inferred, sometimes because not all the steps are part of an instruction, sometimes parts of an instruction are forgotten or not even read, and sometimes because certain steps are part of an error-recovery procedure not part of the original task representation. In such a situation missing steps have to be inferred or guessed. In order to be able to successfully infer or guess steps, we have to again augment our representation of steps to also include what the purpose of the step is. The tea example should be augmented to:

1. [empty kettle] put water in kettle [kettle with cold water]
2. [kettle with cold water] put kettle on stove [water boils]
3. [empty teapot] put leaves in teapot [leaves in teapot]
4. [water boils and leaves are in teapot] pour water in teapot [have tea]

Given this representation it is possible to deduce from the goal of having tea which steps have to be taken. Although this is not necessary in the case of a complete problem representation where forward reasoning suffices, goal-directed reasoning becomes necessary when some step is missing. For example, if we for some reason have forgotten that putting leaves into the teapot is part of the recipe, we can infer that there is a missing gap between having an empty teapot and a teapot with leaves, in which case it is easy to infer the missing step. In many HCI settings inferring a step is not always possible, but just trying out controls might help you find the right step as long as you know what the step you are searching is supposed to do (e.g., wanting to enter the channel on a VCR but not knowing which control to use).

This representation is very similar to STRIPS operators (Fikes & Nilsson, 1971). The difference lies in the way they are used: the instruction representation is not used to fully plan the path to the goal, but only the upcoming step. The projected outcome of a step is only used to build an expectation, or as a help to fill in missing steps in the case of insufficient knowledge. If the expectation does not match the real world, this may be an indication that the knowledge is incomplete, or overly general, in which case an additional control state may be needed.

Conclusions

This chapter has two messages. The first and practically most important is a modeling recommendation to rethink the way we construct our models, and attempt to decrease the reliance on control states. Although I have used ACT-R as an example, I believe this is a guideline that can and should be used in many more architectures. It will not always be easier than the usual practice of many control states, and some architectural changes may be necessary to make control-state-lean models easier to construct. ACT-R 6 already has some changes that make such models easier. Something that is slightly harder to do with a more loose control scheme is to conclude that the goal has been achieved. In a pure top-down model, the goal is achieved whenever the last step has been done. In a mixed top-down/bottom-up scheme this is harder to do, but at the same time opens a way to explain post-completion errors (Byrne & Bovair, 1997).

The second message is that there is an exciting area of modeling to be explored, where the challenge is to construct models that learn unit tasks the way Card, Moran and Newell (1983) intended it: by discovering the unit task boundaries on the basis of knowledge, experience and maybe to trial-and-error.

References

Anderson, J.R., Bothell, D., Byrne, M., Douglass, D., Lebiere, C. & Qin, Y. (2004). An integrated theory of mind. *Psychological Review, 111* (4), 1036-1060.

Anderson, J.R., Taatgen, N.A. & Byrne, M.D. (in press). Learning to Achieve Perfect Time Sharing: Architectural Implications of Hazeltine, Teague, & Ivry (2002). *Journal of Experimental Psychology, Human Perception and Performance*.

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

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

Card, K.C., Moran, T.P. and 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 *Proceedings of the nineteenth annual conference of the cognitive science society* (pp. 107-112). Hillsdale, NJ: Erlbaum.

Clark, A. (1997). *Being there: putting brain, body and world together again*. Cambridge, MA: MIT Press.

Dreyfus, H. (1979). *What computers can't so: a critique of artificial reason*. New York: Harper and Row.

Fikes, R.E. & Nilsson, N.J. (1971). STRIPS: A New approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2,* 189-208.

Gray, W.D. (2000). The nature and processing of errors in interactive behavior. *Cognitive Science, 24*(2), 205-248.

Gray, W.D. & Fu, W.T. (2004). Soft constraints in interactive behavior: the case of ignoring perfect knowledge in-the-world for imperfect knowledge in-the-head. *Cognitive Science, 28*(3), 359-382.

Hazeltine, E., Teague, D. & Ivry, R. B. (2002). Simultaneous dual-task performance reveals parallel response selection after practice. *Journal of Experimental Psychology: Human Perception & Performance 28*(3), 527-545.

Hutchins, E. (1995). How a cockpit remembers its speed. *Cognitive Science, 19,* 265-288.

Larkin, J.H. (1989). Display-based problem solving. In D. Klahr & K. Kotovsky (Eds.), *Complex information processing: the impact of Herbert A. Simon* (pp. 319-341). Hillsdale, NJ: Erlbaum.

Larkin, J.H. & Simon, H.A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science, 11,* 65-99.

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

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

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

Taatgen, N.A. & Anderson, J.R. (2002). Why do children learn to say "broke"? A model of learning the past tense without feedback. *Cognition, 86*(2), 123-155.

Taatgen, N.A. & Lee, F.J. (2003). Production Compilation: A simple mechanism to model Complex Skill Acquisition. *Human Factors, 45*(1), 61-76.

Table 1. Two production systems for a choice reaction task: the productions on the first row implement a three-state model, and the productions on the second row implement a one-state model.

(p perceive =goal> isa crt state start ?visual> state free =visual-location> isa visual-location ==> +visual> isa move-attention screen-pos =visual-location =goal> state attend)	(p retrieve-response =goal> isa crt state attend =visual> isa text value =text ==> +retrieval> isa respons-map letter1 =text =goal> state retrieving)	(p press-key =goal> isa crt state retrieving =retrieval> isa respons-map letter2 =key ==> +manual> isa key-press key =key -goal>)
(p perceive =goal> isa crt ?visual> state free =visual-location> isa visual-location ==> +visual> isa move-attention screen-pos =visual-location)	(p retrieve-response =goal> isa crt =visual> isa text value =text ==> +retrieval> isa respons-map letter1 =text)	(p press-key =goal> isa crt =retrieval> isa respons-map letter2 =key ==> +manual> isa key-press key =key -goal>)

Table 2. A declarative representation of the choice reaction task

Step1	Step2	Step3
isa instruction	isa instruction	isa instruction
task crt	task crt	task crt
condition visual-location	condition visual	condition retrieved-map
action attend-visual	action retrieve-map-visual	action press-key-map

Table 3. Some of the productions to interpret declarative instructions

(p retrieve-instruction	(p implement-visual	(p no-visual-location
=goal>	=retrieval>	=goal>
isa goal	isa instruction	isa goal
task =task	condition	task =task
?retrieval>	visual-location	=retrieval>
buffer empty	action attend-visual	isa instruction
state free	?visual>	condition
==>	state free	visual-location
+retrieval>	=visual-location>	?visual>
isa instruction	isa visual-location	state free
task =task)	==>	?visual-location>
	+visual>	buffer empty
	isa move-attention	==>
	screen-pos	+retrieval>
	=visual-location)	isa instruction
		task =task
		:attended nil)

Author Note

Niels A. Taatgen, Department of Psychology (CMU) and Department of Artificial Intelligence (Groningen). This research was supported by ONR grant N00014-04-1-0173 and NASA grant NRA2-38169. Correspondence concerning this article should be addressed to Niels Taatgen, Department of Psychology, Carnegie Mellon University, Pittsburgh, PA 15213. Electronic mail may be sent to taatgen@cmu.edu.

Figure Captions

Figure 1. Subtasks in making coffee with their dependencies. Simplified version from an example by Larkin (1989).

Figure 2. Diagram of the ACT-R architecture

Figure 3. Time diagram of reading “A” on the screen and pressing the “D” key. Each block in the diagram represents the time that a module requires to complete the particular step.

Figure 4. Time diagram of doing an aural-vocal (thick lines) and a visual-manual (thin lines) task at the same time for a killed performer. In this example the two tasks can be interleaved perfectly. From Taatgen (2005).

Figure 5. Three of the rule sets of the CMU-ASP task. The hexagon shows the condition under which the rule set is applicable, and the dotted line shows the order in which the steps are preferred on the basis of activation.

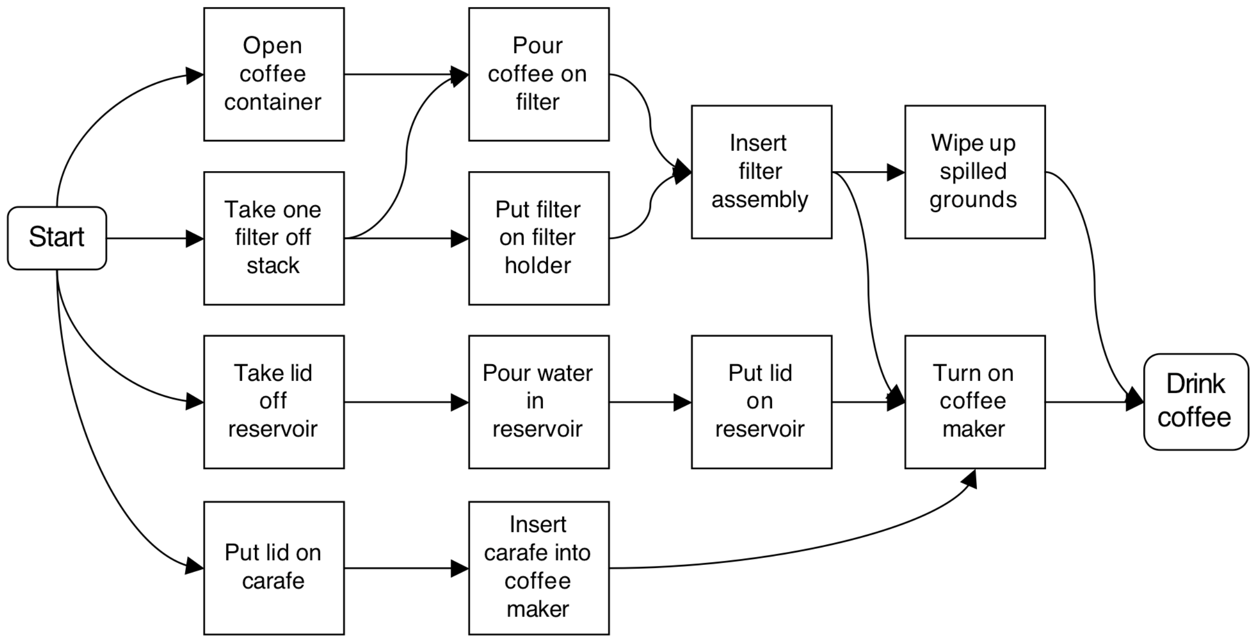
Start
Open coffee container
Take one filter off stack
Take lid off reservoir
Put lid on carafe
Pour coffee on filter
Put filter on filter holder
Pour water in reservoir
Insert carafe into coffee maker
Insert filter assembly
Put lid on reservoir
Wipe up spilled grounds
Turn on coffee maker
Drink coffee

Figure 1.

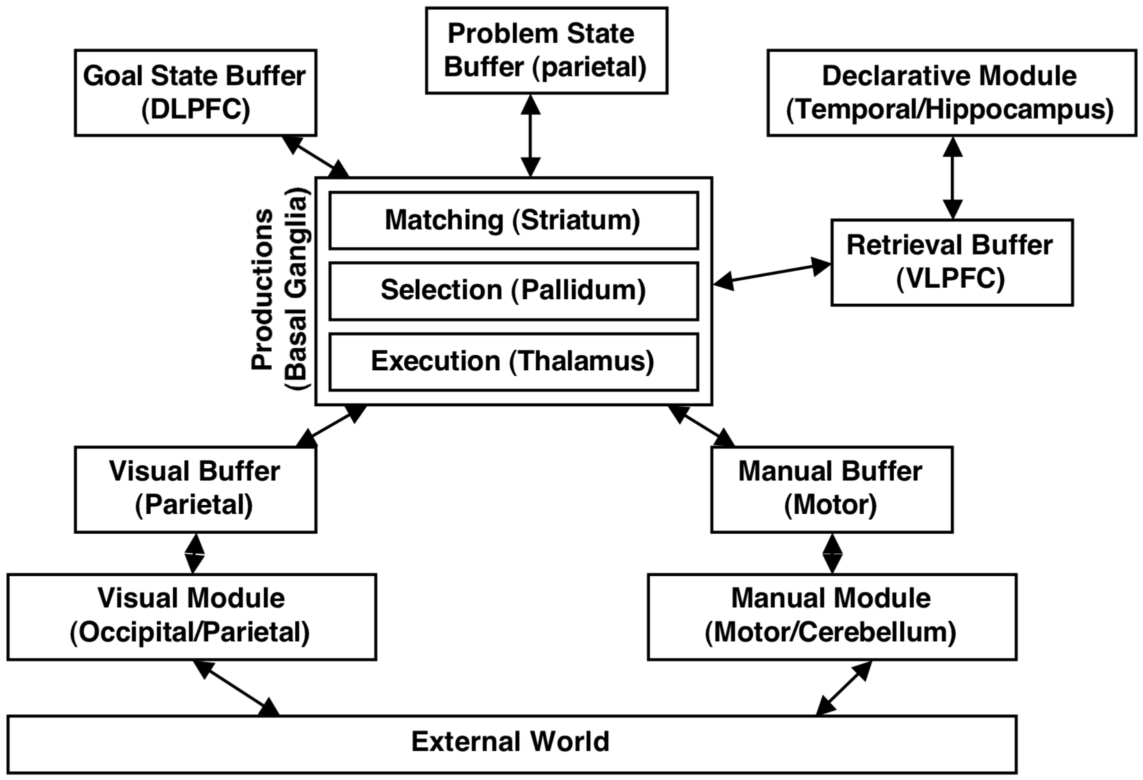
Problem State Buffer (parietal)
Goal State Buffer (DLPFC)
Declarative Module (Temporal/Hippocampus)
Productions (Basal Ganglia)
Matching (Striatum)
Selection (Pallidum)
Execution (Thalamus)
Retrieval Buffer (VLPFC)
Visual Buffer (Parietal)
Manual Buffer (Motor)
Visual Module (Occipital/Parietal)
Manual Module (Motor/Cerebellum)
External World

Figure 2.

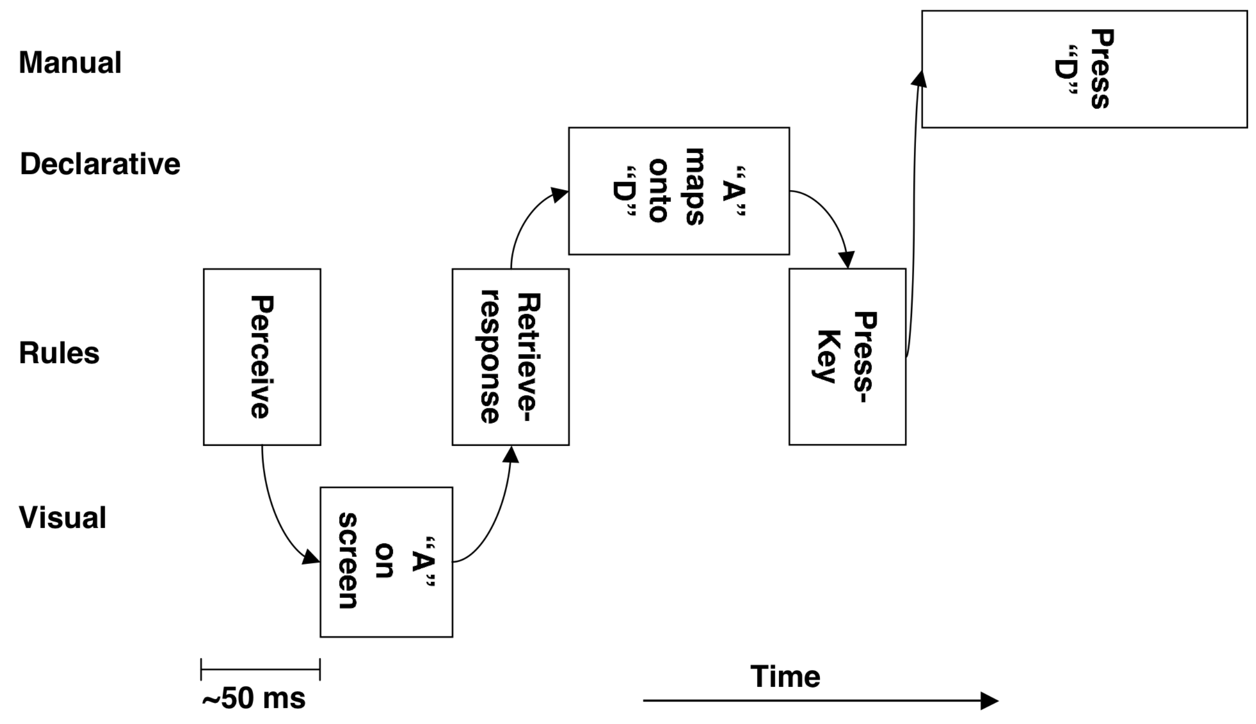
Manual
Declarative
Rules
Visual
Perceive
"A" on screen
Retrieve-response
"A" maps onto "D"
Press-Key
Press "D"
~50 ms
Time

Figure 3.

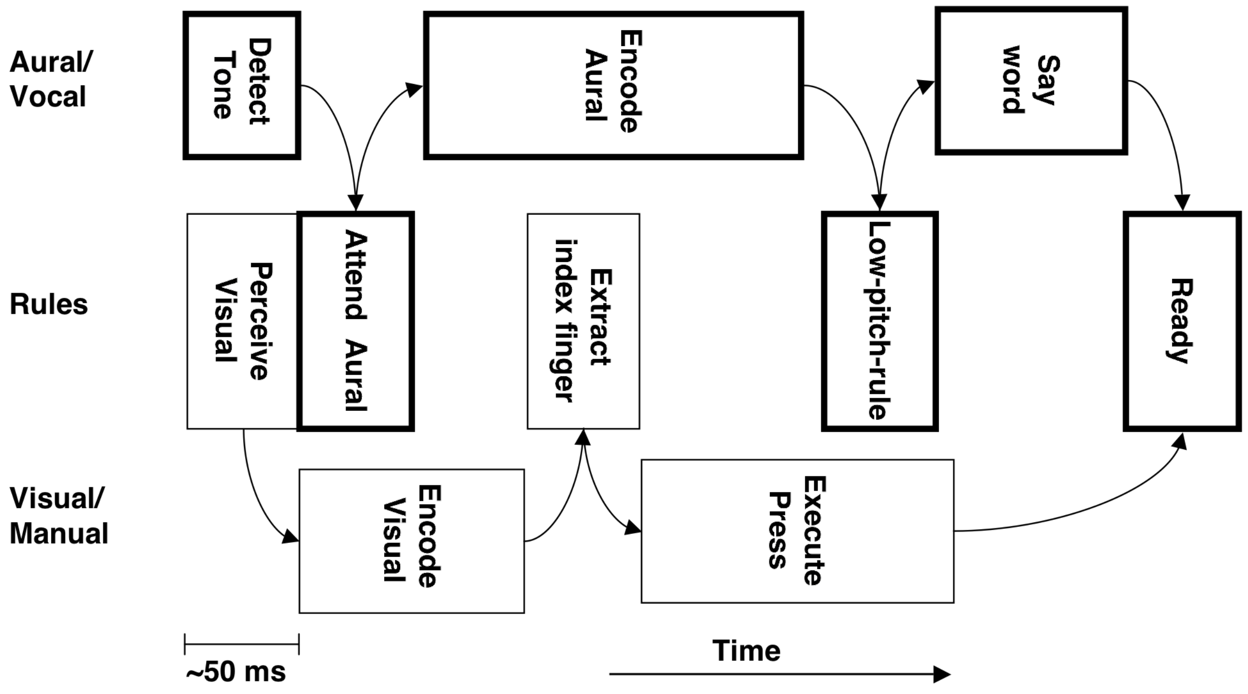
Aural/ Vocal
Rules
Visual/ Manual
Detect Tone
Encode Aural
Say word
Perceive Visual
Attend Aural
Extract index finger
Low-pitch-rule
Ready
Encode Visual
Execute Press
~50 ms
Time

Figure 4.

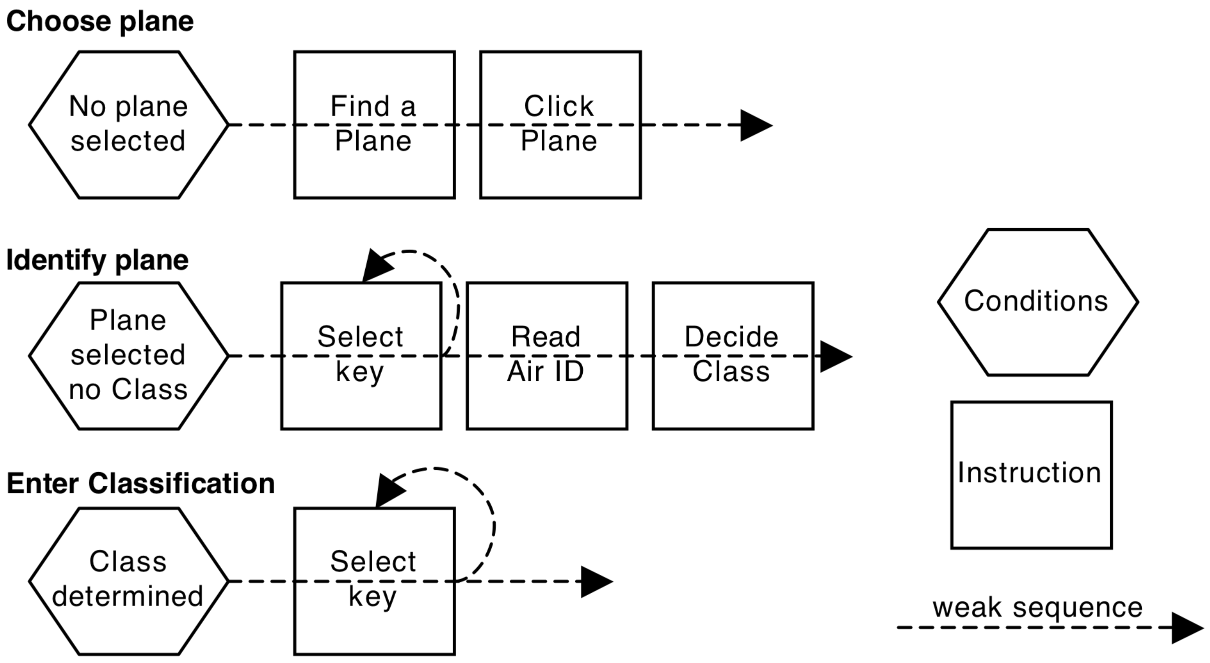
Choose plane
No plane selected
Find a Plane
Click Plane
Identify plane
Plane selected no Class
Select key
Read Air ID
Decide Class
Enter Classification
Class determined
Select key
Conditions
Instruction
weak sequence

Figure 5.