An Account of Model Inspiration, Integration, and Sub-task Validation

Christopher W. Myers
cmyers@cerici.org

Cognitive Engineering Research Institute
5810 S. Sossaman Rd. Ste. 106
Mesa, AZ 85212 USA

Abstract

The promise of reuse is a motivator for, and benefit from, developing cognitive models. Another benefit is the integration of previously developed models into a single model capable of making predictions across different tasks than either of the contributing models could make alone. In the current paper, I explicate the development of a model through the integration of, and inspiration from, two previously published models. The composite was developed for a context different from the constituent models' original contexts, and demonstrates a success of inspiration and integration.

Keywords: model reuse, integration, synthetic teammate

Introduction

The promise of model reuse is a boon to, and motivator for, developing models. A related benefit is to integrate different models into a composite capable of making predictions in different contexts than the contributing models could make alone. To take advantage of these potential benefits, developers of large-scale, complex models must seek out published models for integration rather than *reinventing the wheel*. This paper provides an account of the integration of, and inspiration from, two previously reported models of distinct cognitive processes.

The context for model inspiration and integration is the development of a synthetic teammate (Ball et al., 2009). The constituent models provided portions of the synthetic teammate component responsible for interacting with its task environment, the *task behavior component.*

In the remaining sections of the introduction, I first provide background on the synthetic teammate project. Second, I describe the task and goals specific to the synthetic teammate's task behavior component. Third, I provide results from a task analysis on the goals critical to the task behavior component.

The Synthetic Teammate Project

The Cognitive Engineering Research Institute and the Performance & Learning Models team at the Air Force Research Laboratory are collaborating to develop a synthetic agent capable of coordinating with human teammates to complete an unmanned aerial vehicle (UAV) reconnaissance task. The far-term goal of the project is to reduce the number of human operators in team trainers while maintaining training effectiveness. The near-term goal of this project is to develop a cognitively plausible synthetic teammate within the ACT-R 6 cognitive architecture (Anderson, 2007). Achieving the near-term goal will facilitate accomplishing far-term goals, through the identification of cognitive capacities necessary for operating as a teammate (e.g., memory, language, etc.), and demonstrate how to integrate relevant cognitive capacities.

The synthetic teammate is being developed to operate within a UAV Synthetic Task Environment (UAV-STE; see Cooke & Shope, 2005) used to study teams for the better part of the past decade. In the UAV-STE, teammates coordinate to successfully complete a reconnaissance task. The synthetic teammate has been developed to operate as the UAV Air Vehicle Operator (AVO), and to interact with a photographer that takes pictures of ground targets, or *waypoints*, and a mission planner responsible for planning the UAV's route. Communication among teammates occurs through a text-based instant messaging system.

Four cognitive components have been identified as the basis of the synthetic teammate: 1) language comprehension, 2) language generation and dialog management, 3) situation assessment, and 4) task behavior. The current paper is focused on goals associated with the task behavior component (see Ball et al., 2009, for details on the other components).

Task Goals for the Task Behavior Component

To fly the UAV, the AVO must complete six goals: 1) set the airspeed, 2) altitude, 3) course, 4) waypoint, and 5) send and 6) receive text messages. Of these six, the solution for the first four is covered in the current paper. A typing model associated with the last two goals, sending and receiving messages, is currently being integrated (John, 1996).

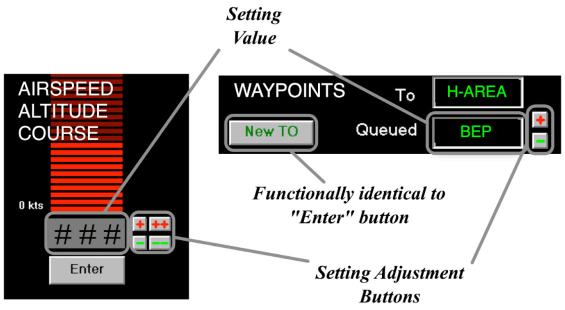

Figure 1. The left box is an example of the interface used to enter the airspeed, altitude, and course. The right box is used for setting waypoints.

The UAV-STE was designed to simulate a team task; consequently the user interface was not designed as a high fidelity representation of any existing UAV system in use by the military. To maneuver the UAV from one location to another, the AVO uses a point-and-click interface to enter settings (see Figure 1). To set the waypoint, the user toggles through a list of 109 alphabetically organized waypoints by pressing the setting adjustment buttons (see Figure 1). Each time an adjustment button is pressed, a new waypoint value is queued (e.g., BEP in Figure 1). The waypoint list operates as a continuous loop (i.e., *A* comes after *Z*). When the user has queued the next waypoint to visit, she presses the "New TO" button, changing "H-AREA" to "BEP" in Figure 1.

There are three *flight parameters* (altitude, course, and airspeed). Each flight parameter has a separate user interface, though they are identical (see Figure 1). To set the flight parameters, the AVO adjusts the setting value by using the small (+ | –) and large (++ | – –) setting adjustment buttons. These buttons have different increments and decrements depending on the setting (see Table 1). Similar to the waypoint list, course values are a continuous loop, returning to 1° after 360°. The airspeed and altitude values are infinite number lines, beginning at 0 and ending at infinity. When the desired setting value is reached, the user presses the "Enter" button to complete the setting goal.

Table 1. Setting adjustment buttons for each task setting goal. The waypoint buttons either increment to the next (+), or decrement (–) to the previous waypoint in an alphabetical list. The other button increments increase or decrease setting values, accordingly.

Task Goals	Large ++ | – –	Small + | –
Airspeed	20 | –20	2 | –2
Altitude	1000 | –1000	100 | –100
Course	10 | –10	1 | –1
Waypoint	Not applicable	1 | –1

There are five differences between setting a waypoint and setting the flight parameters. First, the adjustment buttons for setting the flight parameters have small and large adjustments, whereas there are only small adjustments for setting the waypoint (see Figure 1 and Table 1). Second, there is an "Enter" button for setting the flight parameters and a "New TO" button for setting the waypoint. Although these buttons have different names, there functions are identical. Third, the values of flight parameters are integers, whereas waypoints are strings of numbers and letters (e.g., WP8, BEP). The fourth difference is the addition of the queued value for setting the waypoint, and the fifth and final difference is the spatial arrangements of the user interfaces.

Although there are interface differences between setting flight parameters and waypoints, there is considerable overlap of methods for setting altitudes, courses, airspeeds, and waypoints. In the following section I provide results from a task analysis of the four goals.

Task Analysis Results

A hierarchical GOMS (i.e., goals, operators, methods, and selection rules) analysis was conducted on setting waypoints and flight parameters. The purpose of the analysis was to reveal commonalities and differences between the goals.

The task analysis revealed a consistent three-step subgoal structure across each of the four goals, composed of 1) *obtaining* the desired setting value, 2) c*omparing* the desired setting value against the current value, and 3) *Changing* the current setting value to the desired value. The following methods <*m*> and selection rules <*sr*> are identical across the four goals:

```
Obtain subgoal <sr>:
    Either  Retrieve the desired information from memory
    Or      Request the information from a teammate.
Compare subgoal <m>:
    1. <m>Visually encode one of the adjustment buttons
    2. <m>Move mouse to, and click on, button
    [system-event]:= setting value appears
    3. <m> Visually encode setting value
    4. <sr> IF button adjustment values are unknown,
       THEN retrieve them from memory
    5. <sr> Given the current setting value, desired
       setting value, and adjustment button values, select
       adjustment button
Change subgoal <m>:
    1. <sr> IF mouse is at the selected adjustment button,
       THEN goto <m> 4; ELSE continue
    2. <m> Visually encode button
    3. <m> Move mouse to button
    4. <m> Click mouse
    [system-event]:= setting value changes
    5. <sr> IF not attending to setting value, THEN
       visually encode setting value; ELSE continue
    6. <sr> IF the current setting equals the desired
       setting, THEN visually encode "Enter"/"New TO"
       button and goto change subgoal <m>7; ELSE IF
       large adjustment clicked, THEN goto compare
       subgoal, <sr>5; ELSE goto change subgoal,
       <m>4.
    7. <m> Click mouse–return with goal accomplished.
    [system-event]:= setting value disappears
```

Although setting flight parameters and waypoints follow the same subgoal structure, methods for completing steps in the subgoal methods presented above diverge. The divergence results from different value types between flight parameters and waypoints and the absence of large setting adjustment buttons for setting waypoints. These differences specifically affect methods for completing <*sr*>5 of the *compare* subgoal. In the following section, candidate models for setting the flight parameters and waypoints are selected from the cognitive modeling literature.

Candidate Models

As the science of developing quantitative process models of cognitive activities matures, many models become available with which to take whole cloth or draw inspiration from when tackling large, complex models that must be capable of completing many different tasks. Rather than possibly reinventing the wheel, published models were sought as candidates for integration into the task behavior component of the synthetic teammate. To be a candidate, models had to be compatible with the subgoal methods and selection rules detailed above. The current section covers a strategy selection model (Lovett, 1998) with implications for setting flight parameters, and a letter recall and comparison model (Klahr et al., 1983) with implications for setting waypoints.

Strategy Selection

Lovett (1998) demonstrated that ACT-R's choice mechanism can account for changes in strategy selection with experience from the task environment. Lovett identified two strategies for obtaining a solution in a spatial problem-solving task (i.e., the building-sticks task): *overshoot* or *undershoot*. Generally, the overshoot strategy results in passing the desired state, and then backtracking to it. The undershoot strategy incrementally approaches the desired state without passing it. Strategy selection was based on a strategy's likelihood of success within the environment, such as sets of problems where the overshoot strategy produced a solution a majority of the time and vice versa.

In the building-sticks task, the choice of which strategy to use was not obvious, requiring experience to determine which strategy was most successful. Lovett's model used the production utility mechanism in ACT-R 5 to learn which of the two strategies was best suited for different problem sets. With experience, the model learned to choose a strategy on a proportion of trials that was similar to humans.

Lovett's approach to strategy selection is perfectly suited for selecting between possible strategies for setting flight parameters for two reasons. First, Lovett's model was originally developed in an earlier version of ACT-R. Second, her undershoot and overshoot strategies are similar to strategies that can be brought to bear on setting flight parameters.

When setting a flight parameter, the AVO has four possible adjustment buttons to choose from. From the four options come two strategies: *difference reduction* and *meandering*. The difference reduction strategy involves moving from the current setting to the desired setting, reducing the difference between the two values at each step, and can be achieved *efficiently* or *inefficiently*.

The efficient difference reduction strategy comes as close as possible to the desired setting using the large adjustment buttons, then switching to the small adjustment buttons to reach the desired setting. Indeed, Lovett's overshoot and undershoot strategies are efficient difference reduction strategies.

The inefficient difference reduction strategy involves only using the small adjustment buttons. This strategy will succeed, but in many cases take substantially longer to complete than the efficient difference reduction strategy.

Finally, the meandering strategy is a mix of difference reduction and periodic interventions to randomly select and use a different adjustment button. This strategy will eventually select the desired setting value, but could take months. Hence, only the two difference reduction strategies are considered further.

The efficient difference reduction strategy can be developed as independent undershoot and overshoot strategies, similar to those described by Lovett. Because the structure of the flight parameter setting environment does not contain any bias leading to differential success between an efficient undershoot difference reduction strategy or an efficient overshoot difference reduction strategy, there is little use in developing models of each strategy and letting ACT-R's choice mechanism demonstrate equivalency. Furthermore, the inefficient difference reduction strategy is a "straw man" strategy–participants will arguably use the large increment buttons simply because of their availability.

Letter Recall and Comparison

When setting a waypoint, the AVO can either advance (+) or retreat (–) through the list of waypoints one waypoint at a time. I assumed that participants come to the task with extensive knowledge and experience in the English alphabet. I also assumed that the choice to advance or retreat through waypoints results from bringing the alphabet knowledge to bear on the waypoint setting goal, and looked to Klahr et al. (1983) as a candidate representation of the English alphabet for a model of letter comparison.

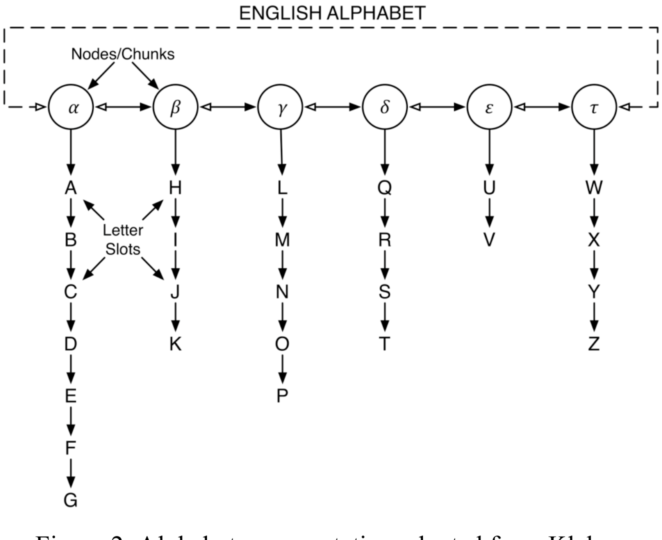

Figure 2. Alphabet representation adapted from Klahr, Chase, and Lovelace (1983). Dashed lines and open arrows represent capabilities added to their model.

In the Klahr et al. (1983) model of letter retrieval and comparison, letters were stored as hierarchical subgroups in a link-node structure (e.g., α to τ in Figure 2). Letters within a node (e.g., D in node α) can only be reached through node *entry points*. Entry points for each node are the first letter of the node (e.g., A for α, H for β, etc., see Figure 2). Node contents are based on empirical evidence of entry point consistency with phrasing in "Twinkle, Twinkle Little Star," used to teach the alphabet (Klahr et al., 1983).

Klahr et al.'s letter retrieval model is a serial, self-terminating search across and within alphabet nodes. Letters and nodes were linked only to their successors. Thus, to backtrack through nodes the previous node must be maintained in working memory.

Klahr et al. validated their model with response time data collected from human participants that were shown letters of the alphabet and asked to respond with the name of the letter that either occurs before or after the probe letter. However, determining whether to advance or retreat through the waypoint list in the UAV-STE is quite different. Rather than responding with an adjacent letter, the model must determine whether the desired waypoint (e.g., BEO) occurs before or after the queued waypoint (BEP in Figure 1). This requires determining if a letter occurs before or after another letter in the alphabet, and these comparisons can occur between and within letter nodes. Even so, the Klahr et al. (1983) model is a good candidate for representing the English alphabet. In the following section I cover the development and integration of the candidate models within ACT-R.

Development & Integration in ACT-R

ACT-R is a computational cognitive architecture for developing cognitive models (Anderson, 2007). In ACT-R, cognition revolves around the interaction between a central production system and several modules. There are modules for vision, motor capabilities, memory, storing the model's intentions for completing the task (i.e., the control state), information retrieved from memory, and a module for storing the mental representation of the task at hand (i.e., the problem state). Each module contains one or more buffers that can store one piece of information, or *chunk*, at a time. Modules are capable of massively parallel computation to obtain chunks. For example, the memory module can retrieve a single chunk from thousands of others and place the chunk into the module's buffer. Module contents are used to guide processing in the central production system.

The central production system is a set of state-action rules that are matched to buffer contents and act on the buffers by removing information from them or adding information to them. Only a single production rule can proceed at a time, and each production rule takes at least 50 ms to complete. The production system acts as a serial bottleneck, as all information passed between the buffers, and interactions with the environment, must go through it.

Developing A Flight Parameter Setting Strategy

The previous section covering Lovett's model of strategy selection revealed that there is not a differential benefit between overshoot and undershoot strategies for setting flight parameters. Not only is there no differential benefit, there are few alternative strategies that would compete in setting flight parameters. Consequently, only the efficient undershoot strategy was developed for setting the flight parameters.

In the ACT-R productions that instantiate <*sr*>5 of the *compare* subgoal, a function was called from a production's action side that selects the appropriate adjustment button given button adjustments for the current flight parameter (e.g., altitude, airspeed, and course) and the current and desired setting values. Button selection was implemented in this fashion to avoid the need of integrating a representation of the number line, integrating models of addition and subtraction, and integrating a model of choosing the appropriate adjustment button based on the button increments and the difference between the desired and current setting values. Hence, the efficient undershoot strategy was perfectly executed by the model when setting the flight parameters. However, the model was not provided knowledge that course values looped back to 1° after passing 360°.

Developing a Waypoint Setting Model

The waypoint adjustment button selection process utilized Klahr et al.'s (1983) model of letter retrieval and comparison. The English alphabet was divided into six *alpha-chunks* that contained letters, instantiating Klahr et al.'s alphabet nodes (see Figure 2). Alpha-chunks were stored in ACT-R's declarative memory, and were based on the Klahr et al. (1983) alphabet division. In addition to letters, the chunk's name and the name of the subsequent alpha-chunk (i.e., the next-node-name slot) were also stored in alpha-chunks. Different from Klahr et al., chunk slots for the chunk name that comes prior to the current chunk in the alphabet (i.e., the previous-node-name slot) and the absolute position of the alphabet chunk in the alphabet (i.e., the position slot with values ranging from 1 to 6) were added to alpha-chunks. The values in the previous-node-name and the next-node-name slots were strings and thus have no effect on memory retrieval in ACT-R.

A two-step process was developed to complete <*sr*>5 of the *compare* subgoal. The process began by comparing the first letter of each waypoint name. If they were equal, subsequent letters were compared until two were different (e.g., O and P from the desired waypoint *BEO* and the queued waypoint *BEP*). At this point the second step began.

The second step began with retrieving alpha-chunks for each of the different letters for comparison (in our example letters O and P). When retrieving alpha-chunks, activation was spread from letters residing in the goal buffer. Thus, alpha-chunks were retrieved independently, without the need to serially traverse the alpha-chunks/nodes until the desired alpha-chunk was reached. This non-serial retrieval of alpha-chunks differs from the Klahr, et al. model, and allows traversing the alphabet nodes in either direction (see open and closed arrows between nodes in Figure 2).

When different alpha-chunks were retrieved, letter comparisons were made using a combination of the previous-node-name, next-node-name, and position slots. However, when retrievals returned the same alpha-chunk, the model had to serially search through the letter slots of the retrieved alpha-chunk until one of the letters was found. To instantiate serial search through slots in the alpha-chunks, *s* x *o* productions were developed, where *s* is the greatest number of letter slots in the alpha-chunk containing the most letters minus one, and *o* is the number of possible outcomes based on comparing two letters. The value for *s* is reduced because if the penultimate slot is reached without finding either of the letters, than the wrong alphabet chunk has been retrieved, searching the last slot becomes useless, and a new retrieval is issued.

The α alpha-chunk had the greatest number of letter slots (i.e., 7), and there were three possible outcomes–the letter from the desired waypoint was reached first in an alpha-chunk, a letter from the queued waypoint was reached first, or the currently checked slot did not contain either letter. Consequently, 6 x 3 = 18 productions were developed to serially search through letter slots of retrieved alphabet chunks. These productions mimicked procedural expertise of iterating through letters within an alpha-chunk. Furthermore, these productions were general enough to apply to any letter comparisons within any of the alpha-chunks.

For example, when the model determines which waypoint occurs alphabetically, BEP or BEO, it determines the first and second letters of the waypoints are identical. Next it determines that O and P are different, and retrieves the γ alpha-chunk. The model then iterates through γ's letter slots, reaching O before P, providing information to the model that BEO comes before BEP in the waypoint list, and to retreat (–) rather than advance (+) through the list.

Although the letter comparison procedure and the declarative structure of the alphabet were based on Klahr et al.'s model, the process differs slightly. For their model to obtain the chunk containing the letter O, it would require retrieving α and β chunks first, then retrieving the γ chunk. Once the γ chunk was retrieved, it would be serially searched for O.

Integration: Sharing Production Rules Across Goals

The methods comprising the subgoal methods and selection rules *obtain*, *compare*, and *change* gleaned from the task analysis suggest that there should be a high proportion of shared production rules to set flight parameters and waypoints when integrating the two models within ACT-R. The similarity in procedures for setting the flight parameters was high, and the only difference was the setting adjustment increments retrieved from declarative memory. Hence, each flight parameter (i.e., airspeed, altitude, and course) shared 100% of its production rules with the other flight parameters. However, production sharing between the setting flight parameters and waypoints was not nearly as high, with a minimum of 32% and a maximum of 44.5%. The minimum value comes from the model not having to serially search through an alpha-chunk, and the maximum value comes from the model having to exhaustively search through the largest alpha-chunk, α.

Both models were successfully integrated into a composite, with a relatively high degree of production rule sharing. In the following section I report the composite's validity.

Composite Model Validation

Model and human participants set the airspeed, course, altitude, and waypoint to determine if the composite model provided valid predictions. Data were collected from three dependent variables: 1) interclick duration, which was operationally defined as the time between clicks beginning after method 2 of the *compare* subgoal, 2) the number of mouse clicks to complete the goals, and 3) the total time to complete the goal. Interclick duration represents temporal dynamics between clicking an adjustment button and determining if the new setting value is the desired setting value (from method 4 through method 6 of the *change* subgoal). The number of clicks and the setting duration reflect the accuracy of the task analysis presented above.

Method

Participants were instructed to set the airspeed, course, altitude, and waypoint 20 times each. There were five human and 10 model participants. Human and model participants interacted with the same environment. Although the model had no knowledge of the course value continuous loop, human participants were instructed that both the waypoint and course setting values were continuous loops.

Base levels for alpha-chunks were set to a high initial value to account for early learning of the alphabet and a lifetime of use. All other ACT-R parameters were set to values necessary for other components of the synthetic teammate. These values were set prior to running the model and remained unchanged. Finally, production compilation and production utility learning were not active during model runs, and the model was reset after setting the flight parameters and waypoint 20 times each.

Twenty randomly selected airspeeds, altitudes, courses and waypoints were randomized for each participant and model run. The model operated as if the desired setting was provided from another teammate through the communication system. Consequently, neither the model nor human participants performed the *obtain* selection rule from the task analysis, presented above.

Results

A comparison between human and model data revealed little deviation between model and human performance, across the dependent variables from each of the four goals (i.e., setting airspeed, course, altitude, and waypoint), *RMSD* = 1.20; r^2 = 0.98.

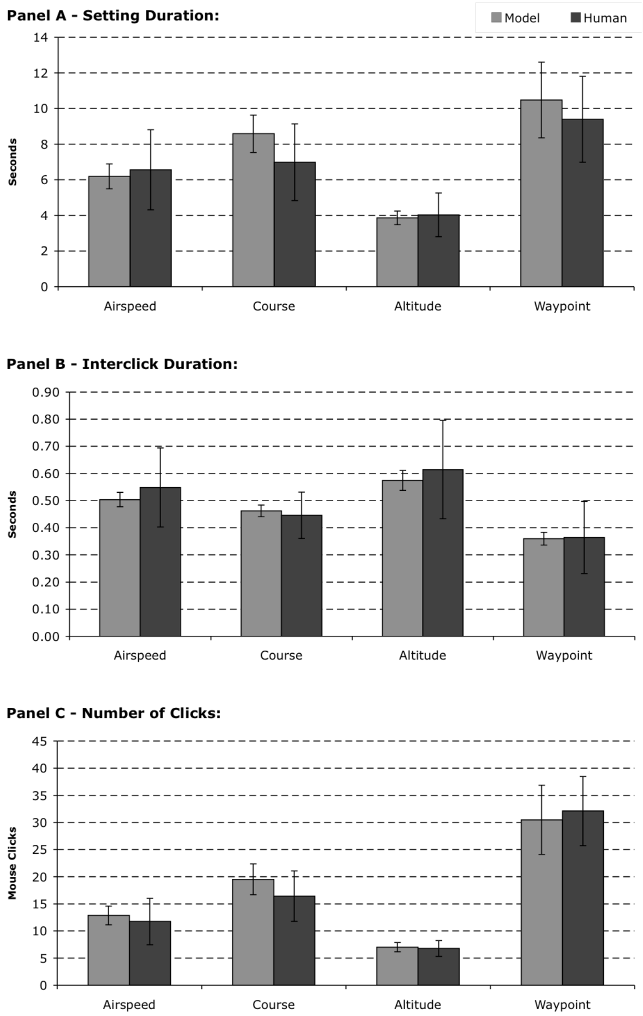

Figure 3. Results from model validation effort for the three dependent variables. Error bars are standard error.

Results indicate a very good fit between the composite model and human data. Interestingly, and not surprisingly, the course flight parameter has the poorest fit to human data, and likely stems from not incorporating knowledge of the setting's continuous loop of setting values.

Discussion and Areas for Improvement

This excellent model fit to human data resulted from performing a detailed task analysis, finding previously published models suitable to perform requisite tasks, and incorporating the models into a composite using a cognitive architecture. Although the model successfully predicts human data, there are clear areas for improvement. First, the selection of the flight parameter setting adjustment buttons is done using a function call external to ACT-R. Incorporating this decision process, while maintaining the model fit to human data is highly desirable. Second, it would be an improvement to enable the model to handle the continuously looping values of the course flight parameter.

The Klahr et al. (1983) model of letter recall and comparison was successfully integrated with other aspects of the synthetic teammate task behavior component. Furthermore, changing Klahr et al.'s serial search across alphabet nodes to a parallel retrieval process using ACT-R's spreading activation mechanism along with the close fit, points to an interesting possible extension to Klahr et al's model. The Lovett (1998) model of choice was less integration–more inspiration. There was also complete sharing of production rules across procedures for setting the different flight parameters, and decent sharing across procedures for setting flight parameters and waypoints. This high degree of production rule reuse reflects success in model integration.

When developing large-scale complex models, such as a synthetic teammate, the model must be capable of completing multiple disparate tasks. Model inspiration and/or integration of existing models provide the developer with the ability to model cognitive activities that may be outside their own area of expertise. The success of the composite model further demonstrates that the development of computational cognitive models has matured enough to draw inspiration from, or integrate, previously published models.

Acknowledgments

The research reported here was supported by grants from the Air Force Office of Scientific Research (AFOSR #FA9550-07-0081) and the Office of Naval Research (ONR #N000140910201) awarded to Dr. Nancy Cooke and Dr. Christopher Myers, respectively.

References

Anderson, J. R. (2007). How Can the Human Mind Occur in the Physical Universe? Oxford: Oxford University Press.

Ball, J. T., Myers, C. W., Heiberg, A., Cooke, N. J., Matessa, M., & Freiman, M. (2009). The Synthetic Teammate Project. Paper presented at the 18th Conference on Behavior Representations in Modeling and Simulation, Sundance, UT.

Cooke, N. J., & Shope, S. M. (2005). Synthetic Task Environments for Teams: CERTT's UAV-STE. In Handbook on Human Factors and Ergonomics Methods (pp. 46-41 - 46-46). Boca Raton: CLC Press, LLC.

John, B. E. (1996). TYPIST: A theory of Performance in Skilled Typing. Human-Computer Interaction, 11(4), 321-355.

Klahr, D., Chase, W. G., & Lovelace, E. A. (1983). Structure and Process in Alphabetic Retrieval. Journal of Experimental Psychology: Learning, Memory, & Cognition, 9(3), 462-477.

Lovett, M. C. (1998). Choice. In J. R. Anderson & C. Lebiere (Eds.), The Atomic Components of Thought (pp. 41). Mahwah: Lawrence Erlbaum Associates.