In Proceedings of the 13th Conference on Behavior Representation in Modeling and Simulation. 04-BRIMS-032. 192-200. Orlando, FL: U. of Central Florida. www.sisostds.org/conference/index.cfm?conf=04brims

CaDaDis: A Tool for Displaying the Behavior of Cognitive Models and Agents

Kevin Tor
Frank E. Ritter
Steven R. Haynes
School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16801-3857
tor@cse.psu.edu, frank.ritter@ist.psu.edu, shaynes@ist.psu.edu

Mark A. Cohen
Business Administration, Computer Science, and Information Technology
Lock Haven University
Lock Haven, PA 17745
mcohen@lhup.edu

Keywords:
Cognitive Model Display, Model Interfaces, Model Behavior, Cognitive Architectures, Agent Behavior

ABSTRACT: *We introduce an extension to an architecture-independent tool (VISTA) for creating displays of cognitive model behavior, the Categorical Data Display (CaDaDis). Our display offers several views of categorical data. It includes a standard Pert Chart showing tasks by category (or resource), a Nonstandard Pert Chart that shows the temporal dependencies, and a Gantt chart that helps show occurrences of agent events along a time line. Perhaps most usefully, it can display categorical and numeric data generated by models as they run. CaDaDis can be used by different cognitive architectures because it has a general interface and creates its displays based on structured messages across a network socket. Its use is illustrated with three, domain-distinct Soar and ACT-R models.*

1. Introduction

Textual traces from cognitive models are often considered unhelpful by observers trying to understand them, and these traces are essentially unintelligible to non-programmers. Modelers and subject matter experts want to know the structure of models as well as how they work [1, 2]. One approach that has been relatively well received is to provide a graphic representation of model's internal processing and behavior. Where this has been done, observers have at least thought they understood the models more, and in some cases have seen and learned new things about their models.

We present a general tool for displaying categorical data generated by cognitive models and AI agents. It can be used by multiple agent and cognitive architectures to display their internal processing. It is based on a no-cost toolkit and implemented in a widely used programming language and should help many models to be understood. The display tool is designed to support the reuse that Newell [3] referred to in his book, in this case by helping models be understood and by being used itself in multiple applications.

We start by describing several displays that have inspired us and provide lessons for our design. We provide example displays created quickly to work with Soar and ACT-R. These displays helped us understand the models we did not write ourselves, and show the types of knowledge that can be gleaned from models using categorical displays of their internal processing. We hope that the reader ends up inspired to create such displays for their model, and that they use our system, CaDaDis.

2. Review of Previous Systems

A wide range of graphical displays have been used by cognitive models. Not every model and not every cognitive architecture has had one, but the displays that have been available appear to help explain models. We examine here several of the displays to show the processing within cognitive models to frame a set of lessons for the design of our tool.

2.1 CPM-GOMS

Gray, John, and Atwood [4] created a task analysis in CPM-GOMS (Critical Path Method/Cognitive-Perceptual-Motor GOMS) for a new and old telephone workstation. They used a modified Pert Chart to represent the sub-tasks and their dependencies. The tasks were aligned by the type of resource they used (voiceinput, visual-input, cognition, and motor output). The modified Pert Chart gave a critical path analysis that allowed estimating the total task time. The displays were created by hand in Microsoft Project, and represented the GOMS model's predictions and interaction.

The displays, while not fully released because they included proprietary data, were used in presentations and papers to explain the process of the model's development, to illustrate the behavior, to debug the models, and to compute the time per task on the new and old workstations (the new workstation was millions of dollars more expensive when the users' task time was included). While the displays were useful, creation of the displays by hand was time consuming and slightly error-prone.

This work illustrated several of the uses of model displays, including the creation and debugging of the model, as well as the important role such displays can play in presenting the model to a variety of audiences. A general display for cognitive models should help with creating models, help with debugging models, and provide displays that can be included in papers and presentations to help explain the models, including conference tutorials.

2.2 APEX

APEX [5] is a tool to evaluate interfaces. It has been extended to have its models and predictions implement the CPM-GOMS architecture [6]. It automatically generates pictorial representations of the actions and their dependencies. These displays have been used extensively in tutorials [7].

APEX showed that automatic displays of the PERT charts could be created, and provided partial answers to the interface needs, including scrolling displays for models with more actions than can fit in a simple display window. This display has been successful enough that a version of it has been included in the ACT-R architecture.

2.3 The DSI and the TSI

The Developmental Soar Interface (DSI) was created to support model creation, debugging, and presentations in Soar 4 [8]. It provided a graphical trace for any Soar model of the problem spaces, states, and operators and their emergent temporal structure. A display of the active states and operators and their problem spaces were used to create a video [9] about Soar. More advanced displays of the operators were used to show which actions matched with human data. These displays showed the cyclical nature of a Soar model [10], and also where learning occurred within a task [11]. An example is shown in Figure 2.1.

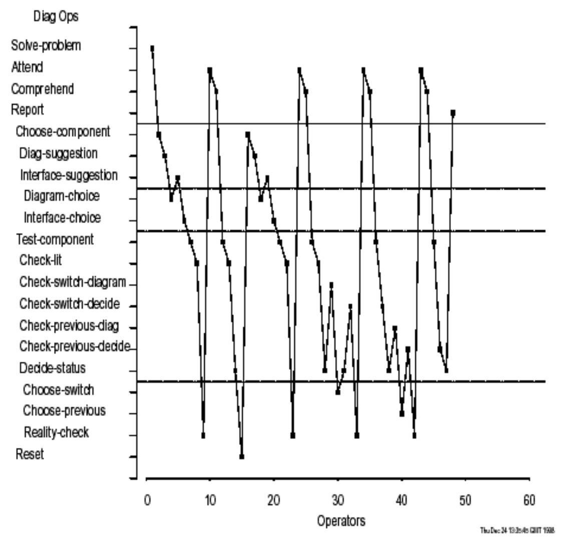

Figure 2.1 Example operator trace from the DSI, a plot generated in S-Plus (Taken from Ritter and Bibby [11]).

Loading and running several models in the DSI showed that Soar models did not have the control structure that had been expoused in Newell's book [3], that of search within problem spaces — the models typically did search across problem spaces, or did little search at all [8, 12].

The Tcl/Tk Soar Interface (TSI) [13] provides multiple views of the working memory and decision processes of a Soar agent, including a semi-graphical trace of the goal stack and the operators in a Soar model. Like the DSI, it works with all models, is a great aid for teaching, and is useful for novice Soar programmers. Its displays, however, have not often been used in presentations, perhaps because they are mainly textual and are somewhat dense graphically.

Both of these systems provide further examples that graphical displays of the models and of their behavior can be helpful to a wide range of users. They also both show that displays can be independent of models.

2.4 Connectionist Modeling Tools

The connectionist model community initially had rather poor interfaces that would print out all of the nodes each epoch (e.g., the early PDP toolkits). But the users could at least see the processing of their model. The graphic displays that came later (e.g., the PDP++ toolkit, [14]) were no doubt a major contributor to the ease of use and uptake of this type of model. Later displays, with their color and nice design even created at times, we suggest, a sense of enjoyment in modeling that has not often been experienced with symbolic models. The availability of displays in this area has to be acknowledged as a partial contributor to their popularity and wide use.

2.5 Display Tools and Communication Media Between Models and Displays

Early displays of models were typically implemented within the same process as the models themselves. The DSI, for example, and APEX, we believe, use this approach. Some users have complained because model displays take time away from the processor running the model, slowing it down [12]. This approach is also less useful where models have been created in systems that do not provide good, fast, easy to use, portable approaches to creating displays and interfaces. Increasingly, Java and Tcl/Tk have been used to create interfaces for models that run in separate processes or on separate machines. These systems have demonstrated that there is value in separating the display from the running model, and the latest versions of the socket utilities have proved more reliable, portable, and easy to use than previous versions [15]. Thus, model displays need a way to create displays and a way for models to communicate with the displays.

Earlier work exploring how to create model displays [13] concluded that it would be difficult to create a completely universal model display. A general interface could be created, but additional displays would likely be required, and a useful approach would be to provide a toolkit and examples that could be customized.

2.6 General Graphic Tools

If only one small display is needed and reuse is not possible nor desired, Remote Method Invocation (RMI) [16] and the Model-View-Controller (MVC) architecture provided by Swing [17] are simpler. Using RMI would require the agent to know how to parse Java messages, and new displays will have to be created. This approach of single displays does not provide a general design or particularly reusable code, and thus does not provide support for accumulating a set of useful displays for models.

2.7 Suggestions for Categorical Data Displays

These display tools reviewed here suggest that including a graphical display of model behavior has several uses, including developing the model, debugging the model, explaining the models to novice programmers as well as interested non-programmers, and providing further insights into the models even for their developers.

Many of the displays reviewed here could be supported by a categorical data display, that is, a display that has time as its X-axis and categories as its Y-axis. These displays would go some way to helping answer the process questions that users ask of models [1, 2].

The most complex objects being displayed are in the CPM-GOMS displays, where Pert- and Gantt-type objects were used, with names, start times, end times, and durations. Many other displays could use simpler graphical objects. With these lessons in mind, we created a general graphical tracing system for cognitive models.

3. Design

We have created a general tool for displaying categorical data generated by cognitive models. The system is implemented in Java and VISTA (which we explain below). We provide a general rational for the design first, and then explain in order of increasing complexity the displays that can be created.

3.1 The Use of Java and VISTA

We chose to create these displays in Java because it is portable and increasingly used to create displays for cognitive models (e.g., ACT-R's new interface, and JACK's eye and hand: [18]). We used the Visualization Toolkit for Agents toolkit (VISTA) [19] because it supports creating displays and a variety of communication channels between the display and cognitive models. VISTA is supported and freely available from Soar Technology, and some of us also used a VISTA tutorial web site (acs.ist.psu.edu/vista) to learn VISTA.

VISTA facilitates the creation of agent visualization applications. This toolkit provides an infrastructure for communication between agents and VISTA enabled applications. Using this communication channel, agents can convey changes to their internal state to a listening VISTA enabled application, which then updates its display to reflect these changes. The VISTA toolkit also provides the ability to record and playback agent activity [20]. Importantly it includes useful Java objects and methods for creating displays of cognitive models (including the ability to parse and deparse objects communicated), as well as a series of examples that can be modified and reused. Using VISTA as the infrastructure for communication between a cognitive architecture and a visualization tool eliminates a significant amount of development time and effort. VISTA proves to be well designed, easy to use, and stable.

VISTA also provides classes to represent some of the major categories of objects based on an analysis of agent systems, including goals, models of other agents, events, time, and so on, along with corresponding sample display methods. These objects are not directly available in Swing or other Java packages. Also, VISTA's connection manager (via SoarComm) offers some remote discovery capabilities, communication utilities, and usability additions that have often been built to support model/interface communications.

3.2 CaDaDis Design

A VISTA window displays the objects traced in the Pert or Gantt view. All views are generated in parallel and a view menu allows the user to toggle between views. Advanced options will be added as this project continues.

Each view consists of a scrollable canvas for graphical objects. Objects represent actions of the agent and are specified as labeled rectangles. The objects each consist of a unique id, an associated code (operator name, etc.), start and end times for execution duration, and an optional list of constraints. Currently, a default value of 50 ms is being used for Soar objects in the Gantt view. This is our assessed time for a decision cycle in the model's run. Constraints are displayed as lines between rectangles. They represent dependencies for the target action. If constraints are not specified, a default line can be drawn from the previous action's rectangle.

A useful design guideline when developing distributed systems is to keep the interface between the client and the server simple. It is important to realize that exposing the visualization client's objects and their methods can tightly couple the client and the cognitive architecture. For complex displays, this level of dependency can be necessary and beneficial, and VISTA supports such complex displays. However, if the designer is not careful strong dependencies can make the system difficult to maintain, and, for simple displays, provide no added benefit. As a result, when designing CaDaDis the classes exposed to the cognitive architecture were minimized.

Our only public datatype is a CaDaDis Object (CDDObject). It can be used to represent any event in the execution of the model whether it is the firing of an operator or the entrance into a new problem space or state. The generality afforded by this implementation provides flexibility to the users of this tool.

3.3 Simple Charts

A simple version of the information can be plotted. This may require adding a bit of code, but would be quite helpful to chart Soar operators in order for a model. Figure 3.1 shows a CaDaDis display of operator firings for the Waterjug demo included with the Soar 8 distribution. The model has four operators (seen on the left as codes) and

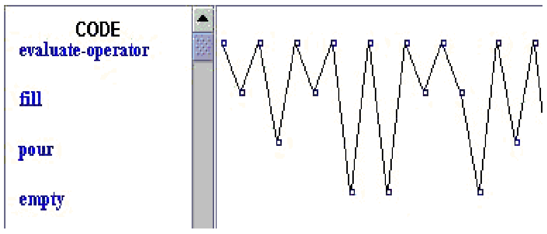

Figure 3.1 A display of a Soar model (Waterjug) with a categorical display of the operators as they are executed.

each point on the right side represents the firing of the corresponding operator. Creating this display required CaDaDis and an included Soar-Tcl interface file. This file contains generic Soar rules used to generate data such as states and operators from any Soar model.

3.4 Pert Charts

A Pert Chart is a display used to do task analysis. It is used to estimate the time to completion for a particular task [21]. Based on dependencies of actions, subtasks can be started earlier if they are independent. Therefore, the end result of the Pert Chart is the realization of the critical path of a task—what is the sequence of actions to complete a task in the least amount of time?

There are two variations on the Pert Chart in CaDaDis. The first is the Standard Pert Chart implementation. The Standard Pert Chart is a simple drawing canvas that contains a set of rectangles representing tasks and arrows representing dependencies between tasks. The standard view is shown in Figure 3.2. Once the model has completed, the view can adjust to show the critical path.

The other is based on John's CPM/GOMS Pert Chart, a Nonstandard Pert Chart, where codes are located on the left-hand side of the window. Objects are drawn in the row of their category. An example is shown in Figure 3.3, it has two drawing canvases. The left canvas is used to hold the codes. The right canvas contains the rectangles and dependency lines.

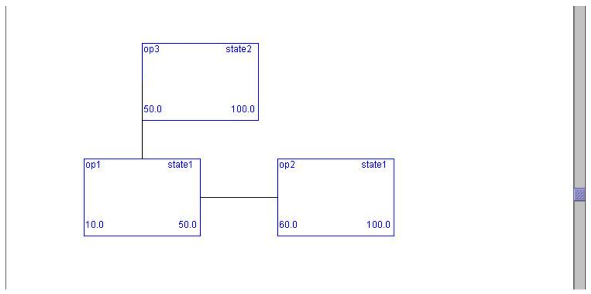

Figure 3.2 A Standard Pert conceptualization.

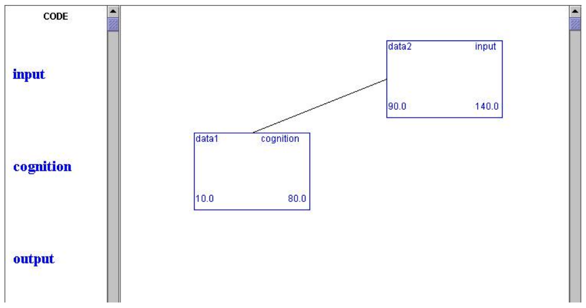

Figure 3.3 A Nonstandard Pert conceptualization.

3.5 Gantt Chart

A Gantt Chart (Figure 3.4) uses a time line to show execution time for a particular action. The codes are on the left side as tasks in the Nonstandard Pert Chart and rectangles are on the right canvas. The size of the rectangle is based on time of execution with respect to the time line.

3.6 Interface Commands

These displays provide enough functionality that users will want to take advantage of their features, but will need support to do so. Table 3.1 lists several of the manipulations users can perform to their displays.

3.7 CaDaDis Messages

VISTA provides built-in data objects. They are the Goal, which can be used to represent a model's goal, and the Milestone, which is an event that a model has completed. The Goal object uses a string to represent the goal's name. The Milestone object uses a string for the milestone name and a float for the time the milestone was reached. For the purposes of CaDaDis, goals and milestones are still too specific. A super object (CDDObject) was created as a peer to VISTA's built-in data objects. It encompasses all of the functionality of a Goal or a Milestone while providing more options supporting other data possibilities.

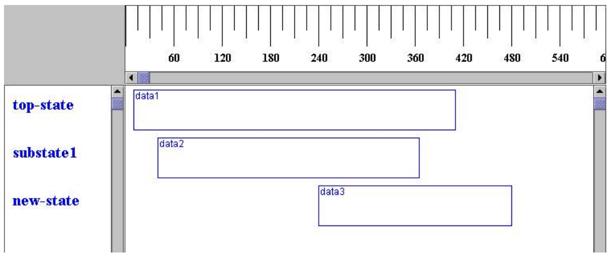

Figure 3.4 A Gantt Chart conceptualization.

Table 3.1 Interface actions that will be (*) or are supported in the display.

- Rescale x and y, and the objects *
- Choose fields to display (i.e., hide some fields to create abstract display, or to highlight the remaining fields) *
- Print the figure
- Save the figure to file, both as a jpeg and semantically
- Change the type of display and provide a best-effort display under the new type
- Provide information about CaDaDis and the type of display
- Provide help functions*
- Set display options, such as autoscroll and absorb new codes (or generate an error)

VISTA objects, like our CDDObject, are created with messages passed on behalf of the model. The format for a CDDObject message is shown in Table 3.2. This message has optional parts (shown in parentheses) that alter the types of messages created. The message creates a new data object *unique_id*, with *tag* as its common name and *code* as the action type, and a *start* start time and *end* end time. The optional *min-start* and *min-end* times are used to determine the model's critical path. The *constraint-list,* when provided, determines how action dependencies are drawn.

Table 3.2 CaDaDis messages for object creation.

```
root create T CDDObject
            S unique_id    S tag
            S code
            F start       (F min-start)
            F end         (F min-end)
             (S constraint_list)
```

4. Example Uses

We present three examples to illustrate CaDaDis. The first example displays operator applications. It uses an example Soar model in the Nonstandard Pert View, which is easy to create, the operator names are placed in the left pane and large or small boxes are drawn along the line corresponding to the operator that fires in the right pane. Presented in all the figures are scaled-down versions of the Nonstandard Pert. As a default, dependencies are specified as the box representing the last operator or production that fired. A portion of this display is shown in Figure 4.1.

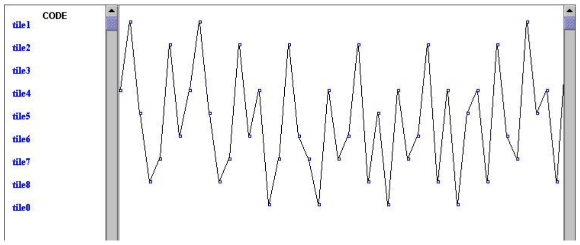

Figure 4.1 Unsolvable 8 puzzle display.

The displays of behavior in the 8-puzzle showed that cyclical behavior can arise in several ways, and that when it is visible those cycles can help understand and troubleshoot models. The 8-puzzle had an initial configuration that was unsolvable. The configuration provided us with a desired behavior. This would have been less apparent in a trace, but the unstable and cyclical pattern of an unsolvable tile configuration was visible in the display. We also found that the direction of moves display (not shown, but N, S, E, and W were the categories), was not useful. This suggests that multiple displays may be required to highlight a model's behavior.

The second example comes from a run of dTank (acs.ist.psu.edu/dTank). dTank is an environment that allows models to play in a simple tank game. In this example, the CaDaDis displays the accessing operators of a simple Soar dTank model. Once again, dependencies are specified as the previously entered state.

Our dTank trace (Figure 4.2) comes from a basic tank agent. This display contains the portion of the run where the model has just found an opponent to attack. The horizontal portion of the trace represents the idle period of the tank waiting for something to come into view. The first set of spikes show the recognition of the opponent. The waiting period is waiting for more perceptions to come in. The decision to attack is made and shown with the subsequent spikes in the trace. This is a nice illustration of agent behavior in the dTank environment.

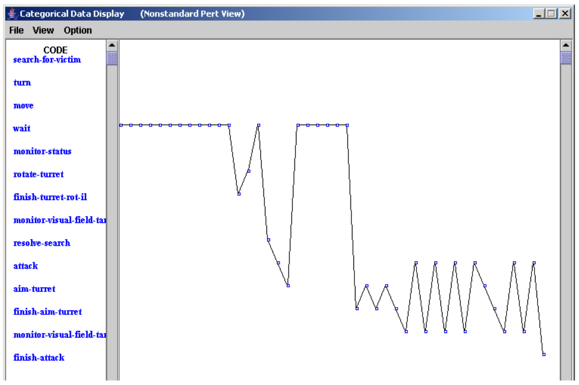

Figure 4.2 A selection an agent trace in dTank.

CaDaDis can be used by multiple cognitive architectures thanks to the architecture independence of VISTA. The final example is from a model written in ACT-R. It simply pulls production names from the model as they fire. The instance of a production firing is represented by a dot in the graph. The ACT-R model traced in Figure 4.3 is a Serial Subtraction model being revised by Andrew Reifers [22]. The model starts with a four-digit number and continually subtracts seven. This behavior is naturally cyclical as shown below. The productions consist of checking for the need to borrow, the act of borrowing, and simple subtraction. This portion of the trace illustrates some cycles where the subtraction was simple but ends with a need to borrow from the hundreds position, hence the downward spike. This display uses CaDaDis and a lisp file addition to ACT-R (now included with CaDaDis).

5. Conclusions

In addition to providing a useful tool for understanding cognitive models and providing a documented and reusable display in VISTA, creating CaDaDis provides several lessons for cognitive modeling and behavior representation. These lessons can be grouped into lessons about our CaDaDis system and the models examined, about VISTA as a tool, and for this process. We take them up in turn.

5.1 Lessons about CaDaDis

Preliminary results show that CaDaDis is successful in showing model behavior. It can create unique displays showing information with more clarity than textual traces. It provides nice displays of model activity in two different cognitive architectures. Furthermore, it can prove useful in debugging cognitive models by analyzing rule usage, whether certain operators fire, and so on.

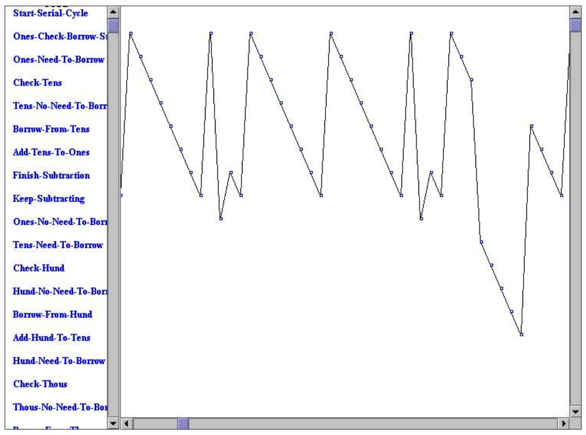

Figure 4.3 Portion of ACT-R Serial Subtraction Model.

The CaDaDis displays included here have become easy to create because creating new displays can be done by modifying previous examples. The displays work with multiple architectures and provide multiple views. They need to be expanded to include more of the functionality in Table 3.1.

To realize their potential, they need to be reused by others. To this end we have put these displays on the VISTA tutorial web site (acs.ist.psu.edu/vista). The displays here created in CaDaDis suggest that the ability to help models explain their behavior can be significantly enhanced with CaDaDis and the VISTA toolkit.

For many applications of VISTA, CaDaDis can serve as a library, providing much of the additional graphics systems will need. For systems that need more complex graphics, or that need to modify the displays we have created, the source code for the graphic displays that make up CaDaDis have been designed to be extended and are documented (acs.ist.psu.edu/CaDaDis). We thus find that we will need libraries of graphical displays for cognitive models as much as we need libraries of model components. It may be possible, however, that interfaces are more reusable.

5.2 Lessons about VISTA

We found that Vista needed to be extended to suit this project. The source code and manual were available, so we saw how to do this. VISTA does provide a base library for 2d-drawing that can be taken advantage of. However, in the example application included in the VISTA distribution, there were no classes related to the specific categorical display objects required for CaDaDis.

The creation of CaDaDis also provides a chance to reflect on the VISTA toolkit as it is used externally. We found that VISTA requires some overhead to learn and to create displays, but provides a worthwhile framework for creating these displays and provides a productive approach to reuse.

In summary, VISTA is a well-designed, easy to use, and very powerful tool. Its use can reduce the development time of agent visualization tools and allow the developer to concentrate on the domain specifics of the application—as opposed to the communication infrastructure.

5.3 Reuse in Cognitive Models and Agents

The reuse of these displays with ACT-R and Soar suggest that the first major reuse of cognitive modeling and agent behavior may be in interface design and not in the knowledge. This might not be that surprising, given that the interface code looks more like the code that gets reused now. Interfaces make up about 50% of most systems [23]. If this is true, which we believe it can be for cognitive models and agents using CaDaDis and VISTA, this is a very worthwhile result.

6. Acknowledgments

This work was supported by the Office of Naval Research through a subcontract from Soar Technology, contract VISTA03-1. Glenn Taylor, Al Wallace, Isaac Councill, Andrew Reifers, and Bill Stevenson provided useful suggestions about the design and implementation of these displays and about this presentation here.

7. References

[1] Avraamides, M., & Ritter, F. E. Using multidisciplinary expert evaluations to test and improve cognitive model interfaces. In *Proceedings of the 11th CGF Conference*. 553-562, 02-CGF-100. Orlando, FL: U. of Central Florida 2002.

[2] Councill, I. G., Haynes, S. R., & Ritter, F. E. Explaining Soar: Analysis of existing tools and user information requirements. In F. Detje, D. Doerner, & H. Schaub (Eds.), *Proceedings of the 5th International Conference on Cognitive Modeling*. 63-68. Bamberg, Germany: 2003.

[3] Newell, A. *Unified theories of cognition*. Cambridge, MA: Harvard University Press 1990.

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

[5] Freed, M., & Remington, R. Making human-machine system simulation a practical engineering tool: An APEX overview. In N. Taatgen & J. Aasman (Eds.), *Proceedings of the 3rd International Conference on Cognitive Modelling*. 110-117. Veenendaal, NL: Universal Press 2000.

[6] John, B., Vera, A., Matessa, M., Freed, M., & Remington, R. Automating CPM-GOMS. In *Proceedings of the CHI'02 Conference on Human Factors in Computer Systems*. 147-154. New York, NY: ACM 2002.

[7] Remington, R., John, B., Matessa, M., Vera, A., & Freed, M. APEX/CPM-GOMS: Modeling human performance in applied HCI domains. In W. D. Gray & C. D. Schunn (Eds.), *Proceedings of the 24th Cognitive Science Conference*. 3. Mahwah, NJ: LEA 2002.

[8] Ritter, F. E., & Larkin, J. H. Using process models to summarize sequences of human actions. *Human-Computer Interaction, 9*(3), 345-383 1994.

[9] Lehman, J. F., Newell, A., Newell, P., Altmann, E., Ritter, F., & McGinnis, T. *The Soar Video*. 11 min. video, The Soar Group, Carnegie-Mellon University. At acs.ist.psu.edu/papers/soar-mov.mpg 1994.

[10] Peck, V. A., & John, B. E. Browser-Soar: A computational model of a highly interactive task. In *Proceedings of CHI '92*. 165-172. New York, NY: ACM 1992.

[11] Ritter, F. E., & Bibby, P. Modeling how and when learning happens in a simple fault-finding task. In *Proceedings of the 4th International Conference on Cognitive Modeling*. 187-192. Mahwah, NJ:LEA 2001.

[12] Ritter, F. E. TBPA: A methodology and software environment for testing process models' sequential predictions with protocols (Tech Report No. CMU-CS-93-101). School of Computer Science, CMU., Pittsburgh, PA 1993.

[13] Ritter, F. E., Jones, R. M., & Baxter, G. D. Reusable models and graphical interfaces: Realising the potential of a unified theory of cognition. In U. Schmid, J. Krems, & F. Wysotzki (Eds.), *Mind modeling - A cognitive science approach to reasoning, learning and discovery*. 83-109. Lengerich, Germany: Pabst Scientific Publishing 1998.

[14] O'Reilly, R.C. & Munakata, Y. *Computational Exploration in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. MIT Press 2000.

[15] Ong, R., & Ritter, F. E. Mechanisms for routinely tying cognitive models to interactive simulations. In *HCI International '95: Poster sessions abridged proceedings*. 84. Osaka, Japan: Dept. of Industrial Engineering, Musashi Institute of Technology 1995.

[16] Arnold, K., J. Gosling, & D. Holmes, *The Java Programming Language*. 3rd ed. Boston: Addison-Wesley. 2000.

[17] Cole, B., et al., *Java Swing*. 2nd ed. Sebastpol, CA: O'Reilly & Associates, Inc. 2003.

[18] Norling, E., & Ritter, F. E. Embodying the JACK agent architecture. In M. Stumptner, D. Corbett, & M. Brooks (Eds.), *AI 2001: Advances in Artificial Intelligence. Proceedings of the 14th Aus. Joint Conference on AI,* 368-366. Berlin: Springer 2001.

[19] Taylor, G., Jones, R. M., Goldstein, M., & Frederiksen, R. VISTA: A generic toolkit for visualizing agent behavior. In *Proceedings of the 11th Computer Generated Forces Conference*. 29-40, 02-CGF-002. Orlando, FL: U. of CF 2002. www.soartech.com/dowloads/VISTA/VISTA.html

[20] Soar Technology Inc., VISTA Developer's Handbook. Soar Technology, Inc 2002.

[21] Harris R. L., *Information Graphic: A Comprehensive Illustrated Reference*. Management Graphics, Atlanta, Georgia, 1996.

[22] Ritter, F. E., Avraamides, M., & Councill, I. G. (2002). An approach for accurately modeling the effects of behavior moderators. In Proceedings of the 11th Computer Generated Forces Conference. 29-40, 02-CGF-002. Orlando, FL: U. of Central Florida.

[23] Myers, B. A., & Rossen, M. B. Survey on user interface programming. In *Proceedings of CHI'92*. 195-202. ACM Press New York 1992.

Author Biographies

KEVIN TOR is a research assistant in the School of Information Sciences and Technology (IST) at Penn State. He has previously obtained an MS in Computer Science from Penn State, and a BS in CS from Seton Hall University. His current research is in creating displays for and optimizing the fit of cognitive models.

FRANK RITTER is one of the founding faculty of the School of IST, an interdisciplinary academic unit at Penn State to study how people process information using technology. He works on the development, application, and methodology of cognitive models, particularly as applied to interfaces and emotions. He is an editorial board member of *Human Factors* and *AISB Journal*. His review (with others) on applying models in synthetic environments was recently published as a HSIAC State of the Art Report.

STEVEN HAYNES is an assistant professor at the School of IST. He researches system design, modeling, and development; human-computer interaction; design rationale; system explanation; and the philosophy of technology. Prior to entering academia he worked at Apple Computer, Adobe Systems, and several smaller technology companies in the US and in Europe.

MARK COHEN is an instructor in the Business Administration, CS and IT Department at Lock Haven University, and a graduate student associated with the Applied Cognitive Science Lab in the School of IST at Penn State. His current research efforts include developing software that simplifies the creation and maintenance of cognitive models. He received an MS in CS from Drexel University and a BS EE from Lafayette College. He has over 10 years of experience developing health care and pharmaceutical software.