Morgan, G. P., Haynes, S., Ritter, F. E., & Cohen, M. (2005). Increasing efficiency of the development of user models. In Proceedings of the 2005 Systems and Information Engineering Design Symposium. Ellen J. Bass, (ed). IEEE and Department of Systems and Information Engineering, University of Virginia: Charlottesville, VA.

`

INCREASING EFFICIENCY OF THE DEVELOPMENT OF USER MODELS

Geoffrey P. Morgan
Steven R. Haynes
Frank E. Ritter

School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802 U.S.A.

Mark A. Cohen

Business Administration, Computer Science, and Information Technology
Lock Haven University
Lock Haven, PA 17745 U.S.A.

ABSTRACT

This paper introduces Herbal, a high-level behavior representation language for creating AI agents and cognitive models. It describes the lessons from other high-level modeling languages that informed the design of Herbal, and that will inform other high-level behavior representation languages. We describe a model built in Herbal to illustrate its use and application. The paper concludes that languages like Herbal can help explain the design intent of intelligent agents and cognitive models, and make them easier to create, modify, and understand. These results appear to be particularly true where the model reuses a lot of its own structures.

1 INTRODUCTION

Despite apparent benefits, user models created in cognitive architectures are rarely found outside of research labs. These benefits are applicable to many commercial sectors, including interface testing and analysis, simulations, and the design of game opponents. Because of steep learning curves, cognitive models are considered too difficult and too expensive for practical use in most commercial endeavors. Expert system solutions are used in many of these areas as a lighter-weight alternative.

Newell (1990) envisioned a culture of model re-use, where models were passed between different research groups and continually evolved over time to study new phenomena. This has proven impractical, because cognitive models currently seem domain-limited and extremely complex. Without access to the original developer, it is often easier to re-write a model to perform the same task than it is to understand borrowed code.

The lack of re-use for cognitive models also contributes to an unfortunate cost-benefit outcome from an observer's stand-point. It seems that costs of collaboration (Brooks 1995) between developers in this area are so high as to be currently effectively insurmountable for most models.

To address these problems, various groups have proposed and developed alternative model development tools that are supposed to make the task easier. Very few of these attempts have been adopted by a large community, or shown significant increase in the utility of cognitive architectures as useful tools. Each of these alternative solutions, however, provides valuable lessons on the road to a useful higher-level language for cognition.

Herbal (originally HRBL: High-level Representation of Behavior Language) is an attempt to make the creation of cognitive models a lighter-weight, easier to initiate solution. By supporting the user and facilitating model reuse, Herbal hopes to make cognitive modeling a more useable modeling technique. Herbal is based on the study of what users ask of a cognitive model (Councill, Haynes, and Ritter, 2003) and is thus intended to be a practical approach to the problem.

Herbal tries to address problems with previous approaches while providing significant utility and a generalist approach to the goal at hand. Herbal, although it only supports Soar at present (see Section 2.1), is designed to eventually support multiple architectures.

This report begins by discussing the theory behind Herbal. It continues by offering a partial review of the literature that influenced Herbal, and presents criteria for a successful HRBL based on this review. Herbal's implementation is then discussed in brief, and finally, we evaluate Herbal in terms of the criteria garnered in the literature review.

2 THE THEORY OF HERBAL

Herbal tries to answer common user questions, including the intent of specific methods, the design rationale, and the known constraints. Herbal is not intended to replace already extant cognitive architectures, instead it compiles

into Soar models. Soar has successfully modeled many psychological and social phenomena and is the only architecture used so far in extremely large-scale simulation of human agents (Ritter et al 2004).

Herbal must be rich enough to represent all of the structures in Soar while hiding that complexity from the user. To do this, Herbal instantiates a set of objects that the user fills in through an IDE (Integrated Development Environment).

When these Herbal objects are mapped into Soar objects through the compiling process, Herbal packages required documentation fields into useful "plain English" explanations with each Soar object based on the latest information provided by the Herbal model developer.

Herbal objects created are based on Soar ontology and theory, and thus an overview of Soar's approach to cognition is discussed in the section that follows.

2.1 Soar's Problem Space Computational Model (PSCM)

Soar is a proposed Unified Theory of Cognition (Newell, 1990) realized as a production system (Laird, Newell, & Rosenbloom, 1987). It uses the problem space computation model (Newell, et al 1991) to organize its declarative (facts) and procedural (actions) knowledge. Soar and the PSCM assume that human cognition is goal-oriented. Soar defines all activity that relates to one goal as belonging to the same problem-space. A problem-space is composed of one or more states and operators, and is resolved when the current state matches a set of criterion established by the developer.

The Soar agent evaluates the current state, proposes possible actions, chooses an action, and then implements that action, creating a new state. Possible action proposals as well as action-implementations are established by the developer.

In choosing an action (an operator to apply), Soar must choose one and only one action. Each proposed operator can be encoded with different preferences. A preference is, in effect, the amount of consideration that action should receive. Possible preferences include best, worst, indifferent, acceptable, and unacceptable. Proposals are also based on the state, and only apply when they are suitable to apply. However, if Soar cannot choose an operator, it declares an impasse and will create a problem-space to resolve the impasse. Previously inapplicable productions, meant to resolve impasses, can now fire and resolve to a single action. There are other forms of impasses, but they are resolved similarly.

When interacting with outside systems, Soar must use an interpreter to implement actions and gather state changes. Otherwise, Soar makes changes to its own working memory structure and re-evaluates the state based on that structure.

Soar repeats this cycle until an acceptable end-state is achieved. Then, either a new goal is proposed (continuing the cycle) or the system halts.

2.2 Mapping Herbal Objects to the PSCM

Herbal (Cohen & Ritter, 2004) implements an ontology of thirteen classes, nine of which are instantiated as objects. These nine object classes, in descending order of granularity are: Model, State, Elaboration, Operator, Impasse, Action, Condition, TopState, and WMObject (Working Memory Object). Each Herbal object carries with it unique explanation responsibilities (Haynes, Councill, and Ritter, 2004; Haynes, Ritter, Councill, and Cohen Submitted) and is used to compile into working Soar code.

The Model class carries the model's meta-information, including its author, its last revision date, its rationale, constraints, and purpose. This high-level information is useful to understand what the designer intended and what problems the designer is already aware of with the model. This object is used to create a documentation header at the beginning of the compiled Soar file which carries this information.

The Herbal developer uses the State class to explicitly resolve foreseen impasses. The State object, in essence, defines the sub-goal of a particular Soar problem-space. Multiple elaborations and operators can fire during the resolution of these sub-states. In Herbal, States must claim particular elaborations and operators for them to be able to fire within that state.

The Elaboration class is used to define 'special' instances that cannot be resolved through the proposal and application of operators. A common example is using an elaboration to halt the Soar system once the desired goal-state has been reached. They are composed of Herbal Condition and Action objects.

Operators are also composed of Condition and Action objects. Both operators and elaborations can have multiple conditions and are allowed to have multiple actions as well. Since many operators have very similar sets of conditions, this reuse of condition objects becomes very useful.

The Impasse class is used to handle foreseen impasse events. It requires that the impasse type be chosen, and a state chosen to handle that impasse.

The Action class defines the application side of Soar apply rules. Documentation slots allow the developer to define what the operator does in plain-English, which is included with each operator and elaboration that includes that action.

The Condition Object defines the conditional side of Soar proposal and application rules. As with Action, documentation slots allows the developer to define the condition simply, and that is packaged with each operator and elaboration that includes the action. Together, Action

and Condition create most of the explanation utility of Herbal.

It should be noted that Soar rules often come in pairs. A proposal rule is used to suggest an action. If that proposal rule is accepted, an apply rule fires, assuming that all the conditions initially true for the proposal phase still hold. Rules are withdrawn as soon as they are no longer applicable.

The TopState class creates the initial state and defines the original problem-space (and usually domain) of the model. In typical Herbal practice, most elaborations and operators are tied to the TopState object, as are WMObjects.

WMObject defines domain-specific memory structures for a particular Soar model. These structures typically define the state of the world for the purpose of the Condition and Action objects. If a model interacts with an outside system, it can rely on input from that system and carry no working memory structure, or it can build working memory and make decisions without relying on the outside system.

3 OTHER WORK THAT INFLUENCED HERBAL

Although driven by the need for explanations to cut costs of collaboration and make models easier to understand, previous and concurrent work influenced the design, goals, and development of Herbal. Each of these approached the problem of making high-fidelity user models easier to build from a different perspective, and each offers lessons.

3.1 Taql

Gregg Yost (1993) developed Taql to harness the power of method-based tools for use in the development of Soar models. His approach centered on unifying the developer's understanding of the PSCM with the language used to define it. He contended that standard Soar merely implemented the PSCM, which led to confusion when developer expectations and actual execution results did not align.

Taql is, in essence, an alternative language for Soar. It is proven to create Soar models in less time than does unaugmented Soar (Yost, 1993). It is also more efficient than other method-based expert systems of its time. Yet, Taql was never widely adopted by the Soar community.

Taql proved to have been even harder for most developers to parse than native Soar. It has a larger and more complex grammar than native Soar or even C (Ritter, 1992), which corresponds to a steep learning curve..

Similar to Taql, Herbal also re-conceptualizes the PSCM for the ease of the developer. Unlike Taql, it does not propose a complex language to replace native Soar. Instead, it integrates Soar code into smaller understandable chunks for the developer through an IDE.

3.2 G2A

G2A (St. Amant, Freed, & Ritter, 2005) is current work and uses an already existing language, GOMSL (Kieras, 1996), to create ACT-R productions. GOMSL is a tool designed to create paper models of the usability of interfaces. It is similar to, but more complex than the Keystroke Level Model (Card, Moran, & Newell, 1980) or GOMS (Kieras, 1988). ACT-R (Anderson & Lebiere, 1998) is a cognitive architecture like Soar, although it has been used primarily to define and understand psychological phenomena.

G2A is effective at creating high fidelity simulations of user-interface tasks. The G2A compiler automates the creation of ACT-R models using GOMS specification with similar accuracy to hand-built ACT-R models. Although not tested fully, the simple comparison of two programmers showed enormous gains in productivity (from weeks of development to hours). This is because GOMSL is not as complex as ACT-R, and thus has a correspondingly simpler grammar.

At this time, it is uncertain if G2A's application extends beyond tasks heavily reliant on user-interfaces. Also, work with G2A has raised significant questions over the roles of compilers. Because GOMSL is so much simpler than ACT-R, G2A's compiler makes choices on how to interpret GOMSL actions to create ACT-R productions. Herbal avoids this issue by retaining a grammar that should provide all of Soar's functionality.

3.3 Other systems

Apex (Freed, Matessa, Remington, & Vera, 2003), COGNET (Zachary, Jones, & Taylor, 2002), and COGENT (Cooper & Fox, 1998) all compile into another, more complex, system. All of these approaches seem both easy to use, and easy to learn <?>. However, none has demonstrated the range and breadth of human performance that Soar and ACT-R have developed.

Because Herbal compiles into Soar, it retains the validity of the Soar approach. These other systems imply that the appearance and aesthetics of a development environment matter, as does consideration of the usability and learnability of the chosen model development environment.

3.4 Summary of lessons learned

From the lessons offered in these alternative approaches, the investigators believe that an effective solution to support modeling will probably compile into a lower-level language of a parent system. The lower-level language should be an already established and validated cognitive architecture. The solution should remain as powerful as the parent, while aligning the mental models of developers with the actual practice of creating models. The solution

should not demand even more of developers than the parent. Finally, the solution system should seem easy to use and learn.

To be considered successful, as with previous attempts performance increases should be demonstrated. In addition to these goals, the system should be easy to pick up and use and should also appear this way.

4 HERBAL'S IMPLEMENATION

Herbal is implemented as an extension of Protégé. Protégé is a graphical ontology editor created and maintained by Stanford Medical Informatics ("Protégé", 2004). Herbal is thus "programmed" by defining the model in Protégé. The tool is available for free under the Mozilla Public License (Mozilla, 2004), and can be downloaded from protege.stanford.edu.

A screenshot of the Herbal user interface is displayed in Figure 1.

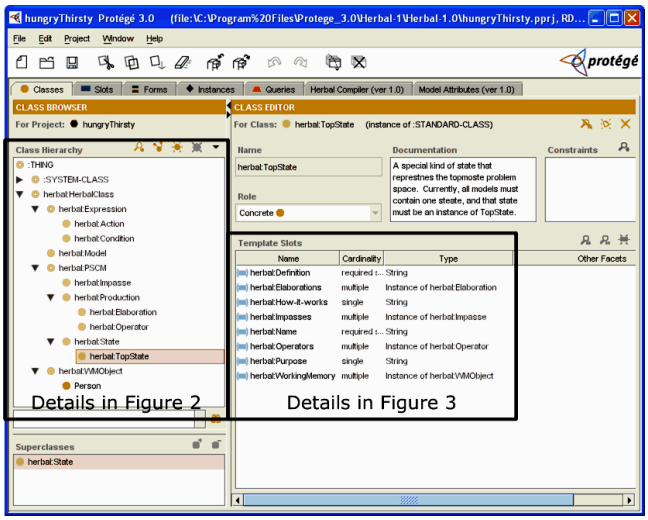

Figure 1. Herbal Screenshot on the Classes Tab

Classes are listed on the right side of the screen, as shown in Figure 2, while specific information about classes can be obtained through the use of several tab panels.

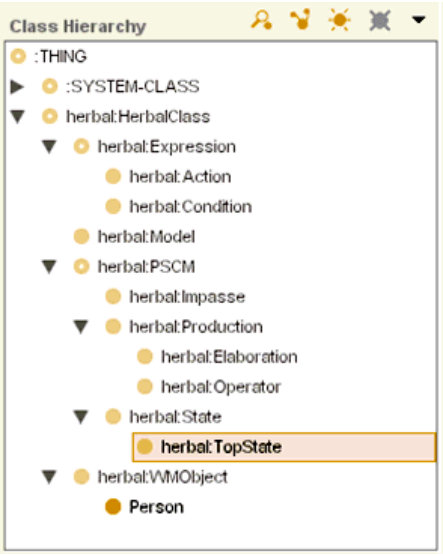

Figure 2. Class region of the Herbal Interface

The class tab provides further information about the class, including variable slots in the class, see Figure 3. The Forms tab describes how Protégé lays out the form for instance definition, allowing the developer to modify formats on the fly. The Instances tab lists all the instantiations of that object in the Herbal model.

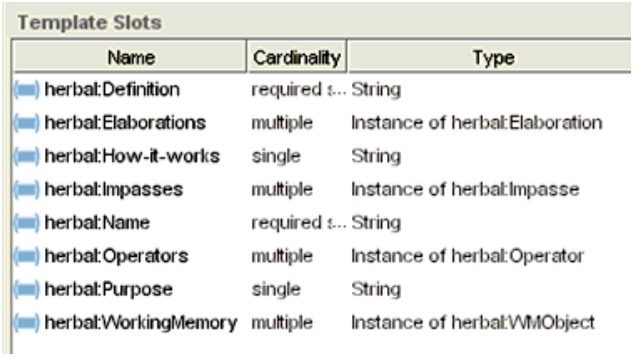

Figure 3. Slot region of the Herbal Interface

Herbal adds two plug-in specific tabs to Protégé. The first, the Herbal Compiler Tab, allows the developer to include outside source-code explicitly in their project. This is particularly useful for models that interact with outside environments, such as dTank, a competitive environment for holistic model comparison (Morgan, Cohen, & Ritter, Accepted). These models often require interface files and with this addition, Herbal can explicitly include them in the final Soar model. This tab lets you compile the Herbal model into Soar code. The second tab, Model Attributes, provides space for developer definition of the model meta-information included in the Model object. Many of these fields are required before Herbal will compile the code, to reinforce the request for useful information.

More information is available about the architecture of Herbal, the Herbal compiler, and the links with Protégé in Cohen, Ritter, and Haynes (Accepted).

5 HERBAL'S EFFECTIVENESS

Herbal is designed to create effective explanations of Soar, and thus should be judged on its ability to explain behavior and intent as well as the criteria summarized in Section 3.4. The investigators examine several aspects of Herbal's effectiveness.

5.1 Can Herbal explain design intent?

To answer this question, the investigators examined Soar operators with and without Herbal's explanation annotations, which encapsulate our best approximation of design intent.

The following snippets are from a model of the Hungry-Thirsty problem, used as an initial problem in the Soar tutorial. The first snippet has had Herbal documentation removed.

```
sp {propose*Drink
    (state <s>
        ^herbal:Name |HungryThirstyWorld|
        ^herbal:WorkingMemory <Person>
        ^flag |drink|)
   -->
   (<s> ^operator <o> + >)
   (<o> ^name |Drink|
        ^Person <Person>)}
```

This second code snippet retains the Herbal documentation.

```
#-----------------------------------------
# Production that proposes operator: Drink
# Operator Definition:
#   Causes a person to drink
# Proposal Conditions:
#   If the drink flag is set
#-----------------------------------------
sp {propose*Drink
    (state <s>
        ^herbal:Name |HungryThirstyWorld|
        ^herbal:WorkingMemory <Person>
        ^flag |drink|)
   -->
   (<s> ^operator <o> + >)
   (<o> ^name |Drink|
        ^Person <Person>)}
```

This represents the best knowledge of design intent available, and does not technically describe all the conditions of this production, because they are assumed or taken care of by the GUI. The actual conditions assumed in this proposal condition are three:

1. There is a state named HungryThirstyWorld
2. In this state, there is a Person object.
3. In this state, an attribute called flag is set to drink.

However, the documentation available, even if inaccurate, presents the reader with the ability to understand the designer's intent when writing the production. This should also enhance a designer's debugging ability, as he can compare what he intended with what he achieved when problems develop.

5.2 Can Herbal explain a running model?

Herbal, itself, cannot explain a running model. However, a tool designed to facilitate the debugging of Herbal models, the Herbal Viewer (Cohen and Ritter, 2003), is capable of providing information on the status of a running Soar model. The Herbal Viewer is a Java-program which requires the developer to import a set of additional methods. These methods send messages to the Herbal Viewer across a socket connection. A file containing all the methods is part of the Herbal Viewer distribution, and importing the file is extremely easy. Please see Figure 4 for a screenshot of Herbal Viewer monitoring a running model.

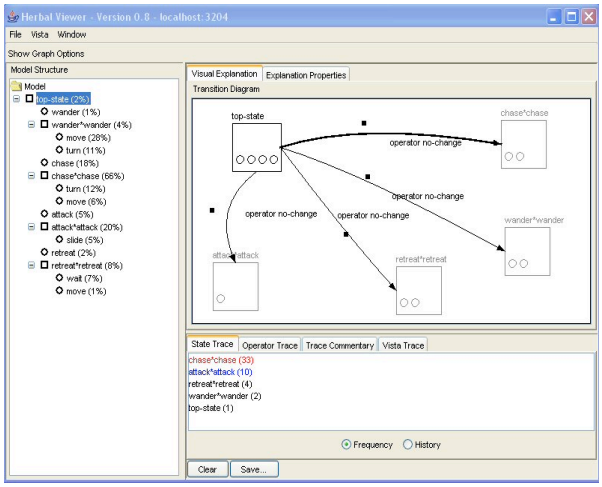

Figure 4. Screenshot of Herbal Viewer of a running model

With this set of views available to the debugging designer, a model can start to explain its behavior, at least to the satisfaction of its designer, which is most useful for debugging purposes. Combined with the static information encapsulated by Herbal, strong inferences can be made about the actions of an agent based on the views provided by Herbal Viewer.

5.3 Is Herbal as powerful as Soar?

Herbal is almost as powerful as Soar, in that it supports nearly all of the mechanisms that Soar provides. The only exception is the weighting of indifference preferences.

This feature is relatively simple, and will be added in the near-future.

Herbal is not, however, as flexible as Soar. Soar provides enough flexibility that different coding traditions have developed over time. Herbal writes Soar productions in a common coding tradition. Theoretically, Herbal could be used to create models that mimic these alternative traditions, but it is does not currently facilitate that process.

5.4 Is Herbal easier to learn than Soar?

Because Herbal is a graphical solution, it does not require the developer to learn a complex grammar as Taql does. This contributes to its ease-of-use. Several classes of undergraduates previously unfamiliar with artificial intelligence have created working Soar models using Herbal. More information on Herbal's use in the classroom is available in Cohen, Ritter, and Haynes (Accepted).

Traditionally, Soar is taught to graduate students or learned independently. The ability of groups of undergraduates to use Herbal speaks highly for its potential. Undergraduates often came to prefer Herbal to Soar as familiarity increased with both systems, even though Herbal requires you to use Soar syntax in areas. This argues that Herbal more closely matches the way that the PSCM is taught to students than Soar's implementation. Therefore, it seems that Herbal is easier to learn than Soar.

5.5 Is Herbal more efficient as a development environment than Soar?

Herbal may prove more efficient as a development environment than un-augmented Soar. At this point, a complete claim cannot be made. It is expected, due to reuse of condition and action statements and the building blocks approach that Herbal offers a developer, that less time will be spent creating and debugging a complex model.

To examine this hypothesis, one of the authors created a model with 29 operators (a rule-pair). The model was created for use in dTank (Councill, Morgan, & Ritter, 2004). dTank is a competitive simulation environment for agents with broad capabilities, allowing for inter-agent communication as well as standard tank behaviors. The model takes advantage of a pre-existing interface between dTank and Soar that ships with the dTank distribution.

The author recorded the time course of the model creation, and computed the marginal cost of producing each rule-pair. The investigators expected that the first rule would be the most expensive, with marginal costs of rule-pairs decreasing as more rules were added. Please see Figure 5.

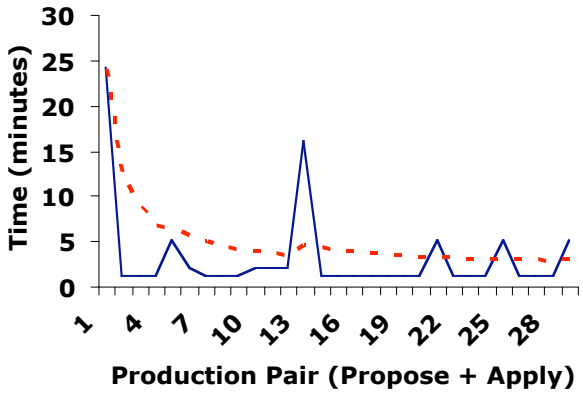

Figure 5. Marginal cost of production-pairs as a function of the number of already existing production pairs. The solid line is the marginal cost. The dashed line is the cumulative average of the marginal costs to that point.

The marginal costs shown in Figure 5 appear to follow the general trend of the investigator's hypothesis; the cumulative average tends to fall as more production-pairs are created. Spikes in time requirements occurred when a new production-pair required more infrastructure, in (mostly) action and condition statements, than what previously exists. After the initial proposal-rule was created, the next most expensive rule was the first rotate-Turret rule, which required a broad set of conditions related to the exact positioning of the turret.

The average of all marginal costs for the model examined was three minutes. This is less than the model development times of graduate students using Taql (Yost 1992). Unlike the effects seen in Taql, the speed-up effects shown above cannot be primarily explained by learning, because the user was familiar with Herbal. Instead, the speed-up appears to relate, as expected, to the increasing reuse of syntactic condition and action statements.

In the near future, the investigators hope to duplicate these results in much larger models. Particularly, the investigators are interested in tasks publicly available as accredited expert systems testbeds, such as the Sisyphus Task documented by Yost and Rothenfluh (1996). However, the current results seem to indicate that Herbal may be as efficient as Taql, a very promising finding.

6 DISCUSSION OF THE HERBAL APPROACH

Herbal has benefited greatly from the examples of previous attempts to simplify the task of making high-fidelity models of users. As an explanation-oriented approach, Herbal will facilitate and thrive in a culture of re-use.

Studies of interfaces often point out that although graphical interfaces appear friendlier, they are often less efficient for expert users than text-based interfaces. This is

because mouse-movements are relatively expensive operations compared to key-typing (Card, Moran and Newell 1980). It is possible that Herbal's performance gains will not be readily apparent or even suffer when expert users are compared. However, frequent re-use of condition and action statements should moderate those effects, which requires expert text manipulation in Soar and then hand-tailoring for each new production.

Although Herbal relies on Soar as the target cognitive architectures, in the future it should be able to compile into any symbolic approach to cognition. This would allow one Herbal model to create multiple architectural models, all of which perform the same task. This would provide a nearly painless method for comparing task performance across architectures, which will help the field move towards eventual architectural unification.

ACKNOWLEDGEMENTS

This work was supported by the Office of Naval Research, contract N00014-03-1-0248. William Stevenson and Joseph Vacchiano provided helpful comments on an earlier drafts.

REFERENCES

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

Brooks, Frederick P. 1995. *The mythical man-month: Essays on software engineering*. 2nd ed: Addison-Wesley.

Card, S.K., T.P. Moran, and A. Newell. 1980. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM* 23 (7):396-410.

Card, Stuart K., Thomas Moran, and A. Newell. 1980. Keystroke-Level Model for user performance time with interactive systems. *Communications of the ACM* 23 (7):396-410.

Cohen, M. A., and F. E. Ritter. 2004. Herbal tutorial. University Park, PA: Applied Cognitive Science Lab.

Cohen, M. A., F. E. Ritter, and S. R. Haynes. Accepted. Herbal: A high-level language and development environment for developing cognitive models in Soar. In *the 14th Conference for Behavioral Representation in Modeling Simulations (BRIMS)*.

Cooper, R., and J. Fox. 1998. COGENT: A visual design environment for cognitive modeling. *Behavior Research Methods, Instruments, and Computers* 30:553-564.

Councill, I.G., S. R. Haynes, and F.E. Ritter. 2003. Explaining Soar: Analysis of existing tools and user information requirements. In *Proceedings of the Fifth International Conference on Cognitive Modeling.*, edited by F. Detje, D. Doerner and H. Schaub. Bamberg, Germany: Universitats-Verlag Bamberg.

Councill, I.G., G.P. Morgan, and F.E. Ritter. 2004. dTank: A competitive environment for distributed agents: Applied Cognitive Science Lab, School of Information Sciences and Technology, Penn State.

Freed, M., M. Matessa, R. Remington, and A. Vera. 2003. How Apex automates CPM-GOMS. In *Proceedings of the Fifth International Conference on Cognitive Modeling*, edited by F. Detje, D. Dörner and H. Schaub. Bamberg, Germany: Universitäts-Verlag Bamberg.

Haynes, S. R., I.G. Councill, and F.E. Ritter. 2004. Responsibility-driven explanation engineering for cognitive models. In *AAAI Workshop on Intelligent Agent Architectures: Combining the Strengths of Software Engineering and Cognitive Systems.* Menlo Park, CA: AAAI Press.

Haynes, S. R., F.E. Ritter, I.G. Councill, and M. A. Cohen. Submitted. Explaining intelligent agents. *Journal of Autonomous Agents and Multi-Agent Systems*.

Keiras, David. 1996. A Guide to GOMS Model Usability Evaluation using NGOMSL. Ann Arbor, Michigan: University of Michigan.

Kieras, D. E. 1988. Towards a practical GOMS model methodology for user interface design. In *Handbook of Human-Computer Interaction*, edited by M. Helander. Amsterdam: North-Holland Elsevier.

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

Morgan, G. P., M. A. Cohen, and F. E. Ritter. Accepted. dTank: An environment for architectural comparisons of competitive agents. In *the 14th Conference for Behavioral Representation in Modeling Simulations (BRIMS)*.

Mozilla. 2004. *Mozilla Public License* 2004. Available from http://www.mozilla.org/MPL/MPL-1.1.html.

Newell, A. 1990. *Unified theories of congition.* Cambridge, MA: Harvard University Press.

Newell, A., G. R. Yost, J. E. Laird, P. S. Rosenbloom, and E. Altmann. 1991. Formulating the problem space computational model. In *Carnegie Mellon Computer Science: A 25-Year commemorative*, edited by R. F. Rashid. Reading, MA: ACM-Press (Addison-Wesley).

Protégé 3.0 (Ontology Editor). Stanford Medical Informatics, Stanford, CA.

Ritter, F. E. 1992. TBPA: A methodology and software environment for testing process models' sequential predictions with protocols. Pittsburgh, PA: Carnegie-Mellon University.

Ritter, F.E. 2003. Soar. In *Encylcopedia of cognitive science*, edited by L. Nadel. London: Nature Publishing Group.

Ritter, F.E., N.R. Shadbolt, D. Elliman, R. Young, F. Gobet, and G.D. Baxter. 2003. *Techniques for model-*

ing human and organizational behaviour in synthetic environments: A supplementary review. Wright-Patterson Air Force Base, OH: Human Systems Information Analysis Center. Available from http://iac.dtic.mil/hsiac/SOARS.htm

St. Amant, R., A. R. Freed, and F.E. Ritter. 2005. Specifying ACT-R models of user interaction with a GOMS language. *Cognitive Systems Research* 6:71-88.

Yost, G. R. 1993. Acquiring knowledge in Soar. *IEEE Expert* 8:26-34.

Yost, G. R., and T. R. Rothenfluh. 1996. Configuring elevator systems. *International Journal of Human-Computer Studies* 44:521-568.

Zachary, W., R. M. Jones, and G. Taylor. 2002. How to communicate to users what is inside a cognitive model. In *Proceedings of the 11th Computer Generated Forces Conference*. Orlando, FL: U. of Central Florida.

AUTHOR BIOGRAPHIES

GEOFFREY P. MORGAN is an undergraduate in the School of Information Sciences and Technology at Pennsylvania State University. He is also a research assistant in the Applied Cognitive Science Lab. His current work focuses on modeling behavioral moderators and studying the actions of teams within contextual units and as coordinated individuals. He can be contacted by e-mail at <gmorgan@psu.edu>

MARK A. 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. He can be contacted by e-mail at <mcohen@lhup.edu>

STEVEN R. 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. His email address is <shaynes@ist.psu.edu>

FRANK E. 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 published in 2003 as a HSIAC State of the Art Report. His email address is <frank.ritter@psu.edu>