High-level Behavior Representation Languages Revisited

Frank E. Ritter, Steven R. Haynes, and **Mark Cohen** (ritter, shaynes, mcohen@ist.psu.edu)

Andrew Howes (HowesA@manchester.ac.uk)

Bonnie John (bonnie.john@gmail.com)

Brad Best (bbest@maad.com) and **Christian Lebiere** (clebiere@maad.com)

Randolph M. Jones (rjones@soartech.com) and **Jacob Crossman** (jcrossman@soartech.com)

Richard L. Lewis (rickl@umich.edu)

Robert St. Amant (stamant@csc.ncsu.edu) and **Sean P. McBride** (sean.patrick.mcbride@gmail.com)

Leon Urbas (leon.urbas@zmms.tu-berlin.de) and **Sandro Leuchter** (sandro.leuchter@iitb.fraunhofer.de)

Alonso Vera (avera@mail.arc.nasa.gov)

Introduction

There has only been a short history of high level languages to model human cognition based on cognitive architectures. TAQL is an early example (Yost, 1993). TAQL showed a large (3x) speed increase over plain Soar, but because it did not support Soar's learning mechanism and because Soar changed soon after its release, TAQL's impact was not as great as its developer probably would have liked.

It is time again to consider high level behavior representation languages. Cognitive models and intelligent agents are becoming more complex and pervasive. This is driving the need for development environments that make it easier to create, share, and reuse cognitive models. Several high level modeling languages have recently been created and several of them are described briefly here. These languages are each different, but they have a common goal of making modeling human data easier to perform. We can now see some generalities and common lessons. By holding this symposium we will identify lessons for the development of these languages as well as for their users. These languages are reviewed briefly in the next section.

Example high-level languages

agimap

agimap (Urbas & Leuchter, 2005) is a tool chain approach that makes cognitive modelling of operator performance in dynamic systems more effective and efficient. agimap was designed for experienced modellers who want to get rid of writing and debugging again and again those parts of the user model that describe behaviour at the interaction level. In contrast to compiler-like-approaches like ACT-Simple (Salvucci & Lee, 2003) or G2A (St. Amant, Freed, & Ritter, 2005). agimap does not leave the modeller with another modelling language but provides an integrated chain of tools that derives the target code from a high-level XML representation of different aspects of the GUI's elements. The chain consists of five distinct components, most of the inner ones not visible to the end user, including: (a) a GUI editor, (b) a GUI description to target architecture compiler, (c) a cognitive architecture / physical world communication layer, (d) a system state to internal GUI representation mapper, and (e) an extendable library of part-task user models to achieve sub-goals at the interaction layer like scanning a set of instruments, reading and monitoring a single instrument, or clicking a single button. Currently, the tool-chain supports part task model creation for ACT-R's perceptual-motor subsystem and its AGI interface. To port agimap to other cognitive architectures, all but the editor component would need to be adapted to the syntax of the target architecture.

To derive a new model, one has to describe the parts of the GUI that are important for the task at hand. For simple GUIs this can be done manually with the GUI Editor that is part of agimap. This results in a model of the human-machine interface coded in XML. For complex GUIs with hundreds of elements per screen (as often found in the domain of process control) it might be reasonable to semi-automate this step. XSLT is then used to transform this XML-representation of the GUI elements to a proper representation for the target cognitive architecture. The transformation generates code for the agimap run-time components (b and c in the component listing above), as well as a representation of the expert users' interface knowledge that would develop during interaction.

The next step is to integrate the appropriate interaction sub-task models with the overall cognitive task model. This might be as simple as "calling" a sub-goal. To get this overall task model right is still the hardest part of user modelling in dynamic systems. However, because the amount of agimap generated code can take up to 50% of the complete user model, agimap helps to concentrate on the essential modelling task.

G2A

G2A (St. Amant et al., 2005) produces ACT-R models from GOMS model descriptions. The GOMS models can

28 feb 2006

contain hierarchical methods, visual and memory stores, and control constructs. G2A allows ACT-R models to be built much more quickly. The data point we have is that it takes an experienced GOMS modeler under an hour to create a dialing model that took an experienced computer scientist graduate student about 50 hours to write in ACT-R (St. Amant et al., 2005). Because GOMS is a more abstract formalism than ACT-R, most GOMS operators can be translated in different ways into ACT-R productions (e.g., a GOMS Look-for operator can be carried out by different visual search strategies in ACT-R).

G2A generates and evaluates alternative ACT-R models by systematically varying the mapping of GOMS operators to ACT-R productions. In experiments with a text-editing task, G2A produced ACT-R models whose predictions are within 5% of the GOMS model predictions. In the same domain, G2A also generates ACT-R models that give better predictions than GOMS, providing good predictions of overall task duration for actual users (within 2%), though the models are less accurate at a detailed level. In a separate experiment with a mouse-driven telephone dialing task, G2A produced models that do a better job of distinguishing between competing interfaces than a Fitts law model or an ACT-R model built by hand.

G2A shows a way forward for cognitive models, that of higher level languages that compile into more detailed specifications. Continuing work on G2A aims at the development of an internal task representation to support translation to other modeling formalisms.

CogTool

CogTool is a tool designed for user interface (UI) designers who have no knowledge or experience in cognitive modeling or programming (John, Prevas, Salvucci, & Koedinger, 2004). By creating an interactive storyboard of a proposed design, then demonstrating tasks on that storyboard, UI designers can obtain predictions of skilled performance time. CogTool combines the theory of the Keystroke-Level Model (KLM, Card, Moran, & Newell, 1980) with a modified ACT-Simple compiler (Salvucci & Lee, 2003) to automatically produce an ACT-R model that is more accurate with respect to user data than previously published KLM results. In addition, the total time to introduce a user to cognitive modeling, train her on CogTool, construct four models, and get execution time predictions was less than half an hour, whereas other novice modelers with far more training took far more time to attain far less accurate predictions with the original KLM and writing ACT-Simple. Furthermore, a modeler skilled in the use of CogTool took an order of magnitude less time to obtain predictions than an ACT-Simple modeler. Recent improvements in the user interface of CogTool decrease this time by an additional 30%.

Unlike the other systems described in this symposium, CogTool does not present a traditional language to the user. In fact, CogTool's philosophy, based on user research, is that its target users, UI designers, will not be willing to use anything that looks like a programming language. Thus, CogTool's "language" is visual and interactive, not something to be typed into a computer. The structure of storyboards is constrained by the CogTool design and frame editors, being translated into the language that defines a device "under the hood" and hidden from the user. Likewise, the modeling "language" that the UI designer sees is comprised of demonstrating steps on the storyboard, which is translated into ACT-R/PM commands. CogTool's capturing of storyboards and demonstrations, however, is generic enough to be translated into many cognitive architectures and is available for other architectures to use. A current effort is to make the CogTool front end connect to IRG. We will discuss the limitations of the current CogTool's expressiveness and plans for future expansion.

Herbal

The Herbal tool set (Cohen, Ritter, & Haynes, 2005) consists of the Herbal high-level behavior representation language, its integrated development environment (IDE), and the Herbal Viewer, which provides explanations of intelligent agent purpose, structure, and behavior. Herbal contributes to this goal by structuring the programming process through the use of explicit class ontology. Although the Herbal tool set does allow designers to create models by directly programming in the Herbal high-level language, the Herbal IDE makes it possible for the designer to interact with this language visually. Like CogTool, the Herbal IDE reduces the learning required by making it a visual task, yet appears to not reduce expressivity, both of which are important.

Another important goal of the Herbal high-level language is to create models that can explain themselves. Herbal contributes to this goal by structuring the programming process through the use of explicit class ontology. The ontology contains classes that represent important Soar model components including: states, operators, elaborations, impasses, conditions, actions, and working memory, all as first-class model objects. The explanation patterns are based on a study of what questions users ask of models (Councill, Haynes, & Ritter, 2003).

Herbal has been used by approximately 40 tutees at the BRIMS conference (Ritter, Morgan, Stevenson, & Cohen, 2005) and by 40 undergraduates at Penn State, where we found a 3x speed up in a simple controlled experiment (Morgan, Cohen, Haynes, & Ritter, 2005). Herbal provides about a 8x code expansion.

Though Herbal is currently designed to produce Soar agents, we intend to continue development of the environment so that it is capable of producing both ACT-R and Jess agents. This multi-platform capability will facilitate sharing design knowledge across three of the largest intelligent systems development communities.

HLSR

HLSR is a High Level Symbolic Representation language for the development of cognitive models and intelligent agents (Jones, Crossman, Lebiere, & Best, 2006). Our work is in the spirit of past research into

cognitive architectures, which provide functional components and data representations for the primary purpose of modeling human behavior. Each cognitive architecture essentially defines an abstract machine together with a language for programming that machine. However, until recently, there has been little effort to identify in a formal way the commonalities across existing cognitive architectures, which would also make it more clear which architectural differences are important from a theoretical point of view (although, see Cooper, Fox, Farringdon, & Shallice, 1996).

Our experience has shown that there are important fundamental and theoretical differences between the most prominent cognitive architectures. At the same time, however, much of the work involved in building specific cognitive models is the same no matter which architecture one is using. This is particularly true for defining high-level knowledge representations, building a structured task analysis, and implementing this with a somewhat standard sense-retrieve-act decision cycle. Part of the intent of this research is to make it feasible for these common modeling activities to be accomplished within a common framework and formal language. This should make it easier to build and maintain models, to explore model variations within a particular cognitive architecture, and to compare models usefully across architectures.

To this end, HLSR defines an abstract formal programming language for cognitive modeling that attempts to generalize the common structures and processes found in existing cognitive architectures. Our approach has been to combine a high-level overview and analysis of a number of architectures for cognition and intelligent agents together with a fine-grained analysis of two of the most prominent cognitive architectures. Our current work involves developing a language and compilers to specify high-level cognitive models and translate them into executable ACT-R and Soar models. This has required us to be extremely careful about managing the theoretical differences and assumptions behind ACT-R and Soar, and generalizing those into a useful abstract framework that can be represented in a formal, high-level language. An additional benefit of this approach is a formalization of common modeling patterns in those architectures and an examination of their practical implications beyond ad hoc modeling practice.

IRG

IRG is motivated by the observation (Howes, Lewis, Vera, & Richardson, 2005; Lewis, Vera, & Howes, 2004; Vera, Howes, McCurdy, & Lewis, 2004) that existing languages for representing routine cognitive tasks (such as GOMS, UAN, and PDL) can fail either because they demand that task competence is described using serial position to determine temporal order (and they are therefore overly restrictive) or because they demand that partial orderings are specified with temporal dependencies and other logical relationships (and they are therefore under-constrained). Howes et al. (2005) propose a novel task description language, called Information-Requirements Grammar (IRG), which is consistent with a

theory of how higher-level task performance is constrained by the information requirements and resource demands of lower-level tasks. We have (Howes et al., 2005; Eng et al., 2006) have demonstrated the use of IRG and show how it replaces serial ordering and temporal dependencies with resource-bound information cascades between architectural information processes.

For example, a GOMS method, or any of its derivatives, for verbally requesting a postcode and entering into a database field imposes an ordered sequence:

```
GOMS: Enter postcode -->
      Step 1: request postcode,
      Step 2: listen for postcode,
      Step 3: select postcode field,
      Step 4: type postcode.
```

In contrast, IRG allows specification of information requirements in capitalized parameters to each subtask. The RHS subtasks are not ordered by position but are constraint by information needs. Therefore the select task can proceed in parallel with the request and listen tasks. The type task must wait for all three to complete.

```
IRG:  enter postcode -->
      request postcode to give CODE,
      listen    for    CODE    to    give
         CODE_REPRESENTATION,
      select    postcode    field    to    give
         SELECTED_FIELD,
      type        CODE_REPRESENTATION        into
         SELECTED_FIELD.
```

The above method is not pseudo-code. It is the form that is used. Large hierarchical grammars can be specified in this fashion.

Discussion and Conclusions

Several high level modeling languages now exist. They are increasingly usable and are able to create models validated by data. They will soon be invaluable modeling and theorizing tools. They now need to be introduced to a wider audience, and with increased use have their development accelerated.

These tools will be increasingly important for modelers—creating models quickly. They will make theories in this area more mutable, maintainable, and creatable. For modeling to become more important in psychology, models will have to become easier to use. This symposium helps disseminate good work in this area, and should help more modelers help develop higher level languages, as well as find higher level languages for their own use.

There remain important open problems and questions about these languages. These questions include:

- What are the research goals of these systems? Are they languages, modeling tools, or both?
- What cognitive architecture(s) are the system built upon? What architectures are easy to use in this way? How do the tools influence the architecture?

- What does the high-level langue look like? (What's it like to use?) Is it faster or more enjoyable, or both, or neither?
- How many models have been built, and how many users have used it? How mature is the language?
- What is the largest model built so far? Does the language scale well?
- Is there any evidence for the predictions of the models the high-level language creates? Are the models produced accurate?
- Is there any evidence for the tool's usability? How do we know the tool is usable outside the lab that developed it?
- What constraints does the language impose on adaptation and learning? More broadly, what types of behavior does the language support modeling?

Acknowledgements and Author Affiliations

References

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

Cohen, M. A., Ritter, F. E., & Haynes, S. R. (2005). Herbal: A high-level language and development environment for developing cognitive models in Soar. In *Proceedings of the 14th Conference on Behavior Representation in Modeling and Simulation*, 133-140. 105-BRIMS-043. Orlando, FL: U. of Central Florida.

Cooper, R., Fox, J., Farringdon, J., & Shallice, T. (1996). A systematic methodology for cognitive modelling. *Artificial Intelligence, 85*, 3-44.

Councill, I. G., Haynes, S. R., & Ritter, F. E. (2003). Explaining Soar: Analysis of existing tools and user information requirements. In *Proceedings of the Fifth International Conference on Cognitive Modeling*, 63-68. Bamberg, Germany: Universitäts-Verlag Bamberg.

Eng, K., Lewis, R. L., Tollinger, I., Chu, A., Howes, A., & Vera, A. (2006). Generating automated predictions of behavior strategically adapted to specific performance objectives. In *Proceedings of ACM Conference on Human Factors in Computing Systems, CHI'06*. New York, NY: ACM.

Howes, A., Lewis, R. L., Vera, A., & Richardson, J. (2005). Information-Requirements Grammar: A theory of the structure of competence for interaction. In *Proceedings of the 27th Annual Meeting of the Cognitive Science Society*, 977-983. Hillsdale, NJ: Lawrence Erlbaum.

John, B. E., Prevas, K., Salvucci, D. D., & Koedinger, K. (2004). Predictive human performance modeling made easy. In *Proceedings of CHI 2004 (Vienna, Austria, April 2004)*, 455-462. New York, NY: ACM.

Jones, R. M., Crossman, J. A. L., Lebiere, C., & Best, B. J. (2006). An abstract language for cognitive modeling. In *Proceedings of the 7th ICCM*. Mahwah, NJ: Lawrence Erlbaum.

Lewis, R. L., Vera, A. H., & Howes, A. H. (2004). A constraint-based approach to understanding the composition of skill. In *Proceedings of the Sixth International Conference on Cognitive Modeling*, 148-153. Mahwah, NJ: Lawrence Erlbaum.

Morgan, G. P., Cohen, M. A., Haynes, S. R., & Ritter, F. E. (2005). Increasing efficiency of the development of user models. In *Proceedings of the IEEE System Information and Engineering Design Symposium*.

Ritter, F. E., Morgan, G. P., Stevenson, W. E., & Cohen, M. A. (2005). A tutorial on Herbal: A high-level language and development environment based on Protégé for developing cognitive models in Soar. In *Proceedings of the 14th Conference on Behavior Representation in Modeling and Simulation*. Orlando, FL: U. of Central Florida.

Salvucci, D. D., & Lee, F. J. (2003). Simple cognitive modeling in a complex cognitive architecture. In *Human Factors in Computing Systems: CHI 2003 Conference Proceedings*, 265-272. New York, NY: ACM.

St. Amant, R., Freed, A. R., & Ritter, F. E. (2005). Specifying ACT-R models of user interaction with a GOMS language. *Cognitive Systems Research, 6*(1), 71-88.

Urbas, L., & Leuchter, S. (2005). Model based analysis and design of human-machine dialogues through displays. *KI – Zeitschrift für künstliche Intelligenz [AI-Journal for AI]*, 45-51.

Vera, A., Howes, A., McCurdy, M., & Lewis, R. L. (2004). A constraint satisfaction approach to predicting skilled interactive performance. In *Proceedings of CHI 2004*, 121-128. ACM Press: New York, NY.

Yost, G. R. (1993). Acquiring knowledge in Soar. *IEEE Expert, 8*(3), 26-34.