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. 177-182. 05-BRIMS-044. Orlando, FL: U. of Central Florida

Herbal: A High-Level Language and Development Environment for Developing Cognitive Models in Soar

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

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

Keywords:
Cognitive Modeling, Development Environments, High-Level Languages, Ontologies, Protégé, Soar

ABSTRACT: *Cognitive architectures are useful to a wide variety of users. To accommodate this wide range of users, and to promote the use of cognitive systems, it is essential that tools such as high-level languages and development environments are created to allow the modeler to focus more on the problem domain, and less on the nuances of a particular architecture. This paper introduces a development environment and high-level behavior representation language called Herbal that represents a step towards creating tools to support a wide range of cognitive model users.*

1. Introduction

Cognitive architectures are useful to a wide variety of users. Computer scientists, psychologists, cognitive scientists, and various domain experts all have uses for cognitive architectures. Unfortunately, designing, implementing, and using cognitive architectures can be a difficult task considering that the background and expertise of this diverse set of users varies from novice to expert (Ritter et al., 2003). In addition, the tasks performed by users of such architectures can vary widely, and can include a rage of different tasks.

Part of the problem stems from the fact that cognitive architectures typically use low-level programming languages to model behavior. Both Soar (Laird, Congdon, & Coulter, 1999) and ACT-R (ACT-R Research Group Department of Psychology Carnegie Mellon University, 2004), for example, use production rules as their primary programming construct.

A wider variety of users can be supported in the task of developing cognitive models using a language that maps more directly to the domain the user is familiar with. For example, if this high level language is built on top of Soar, and closely resembles the theory used by the Soar architecture (Newell, 1990), it would make Soar easier to use, while maintaining the features of Soar that make it interesting, including learning, interaction, and interuptability.

To promote the use of cognitive architectures, it is essential that tools such as high-level languages and development environments are created to allow the modeler to focus more on the problem domain, and less on the implementation nuances of a particular architecture.

Several tools already exist to simplify the development of cognitive architectures. For example, Soar developers have VisualSoar (Laird, 1999), a powerful development environment that simplifies the creation of Soar productions. SoarDoc (Soar Technology Inc., 2004) is a tool for automatically generating HTML documentation directly from Soar source code. ViSoar (Hirst, 1999) is a dialogue-driven interface for generating Soar code automatically, and debugging, and reverse engineering existing Soar productions. For ACT-R there is G2A (St. Amant, Freed, & Ritter, 2005), and other architectures include Integrated Development Environments (IDE) (Busetta, Rönnquist, Hodgson, & Lucas, 1999; Lebiere et al., 2002; Zachary, Jones, & Taylor, 2002). A further partial review of similar tools can be found in (Morgan, Cohen, Haynes, & Ritter, in press)

We propose a cognitive modeling tool that involves the use of a high-level programming language and a compiler. This tool is different because it is designed not only to simplify the development of cognitive models, but also to generate explanations of the running models.

The Herbal development environment introduced in this paper makes Soar cognitive models easier to develop and maintain by using a high-level language, a compiler, and by implementing a theory for generating explanations.

2. Rationale for a High-Level Language

The usability of Soar in applied settings appears to be limited by the unstructured ontology of Soar programs. Developers are required to code Soar models at the production level. The resulting programs are thus formed as a list of condition-action rules with no particular order or arrangement, except that which arises from the coding conventions employed by the responsible developer(s). All information regarding the intended processes of agent cognition and behavior must be inferred by developers while interpreting Soar code at the syntactic level, observing run-time traces or displays (Taylor, Jones, Goldstein, Frederiksen, & Wray, 2002), or communicating directly with the developer.

Even experienced developers can have difficulty understanding Soar code written by others. The lack of explicit process representations coupled with the freedom to adopt widely disparate coding conventions poses significant difficulties for understanding large Soar programs. Developers must actively generate their own mental models of program behavior from the atomic model components. Also, the common structures within a program, such as representations of external objects, are stored in multiple locations, requiring the programmer keep track of objects without the aid of an IDE.

To improve the creation and readability of Soar programs, a high-level language based on an ontology model was created that presents a useful objectification of Soar program components. Developers specify Soar systems at the ontological level and have code generated directly from these high-level design specifications.

3. The Herbal High-Level Language

The main goal of the Herbal high-level language is to create models that can explain themselves (Haynes, Councill, & Ritter, 2004; Haynes, Ritter, Councill, & Cohen, 2005). As part of this process, Herbal makes it easier to program cognitive models. The Herbal high-level language accomplishes this by formalizing the programming process through the use of an explicit ontology of classes (as shown in Figure 2.1). The ontology contains classes that represent concepts such as models, states, operators, elaborations, impasses, conditions, actions, and working memory, all as first-class model objects.

The classes in this ontology also have some basic relationships (Figure 2.2). A state can contain impasses, working memory, operators, elaborations, and other states. In addition, operators and elaborations can contain zero or more conditions and actions.

Programming in the Herbal high-level language involves instantiating objects using these ontological classes. Thus, the process of programming a model is reduced to instantiating objects from a set of fixed classes, instead of coding the classes and structure implicitly in a large set of heterogeneous Soar productions.

When objects are instantiated, the ontology includes additional attributes that allow the developer to describe the intent of the object. This descriptive information is automatically embedded in the generated Soar code as comments when the model is compiled. It will be used in future version of Herbal to generate explanations.

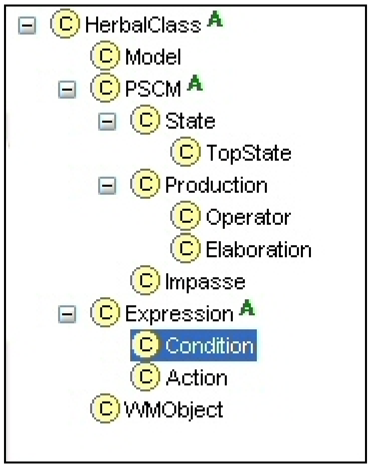

Figure 2.1 Herbal Ontology Class Hierarchy

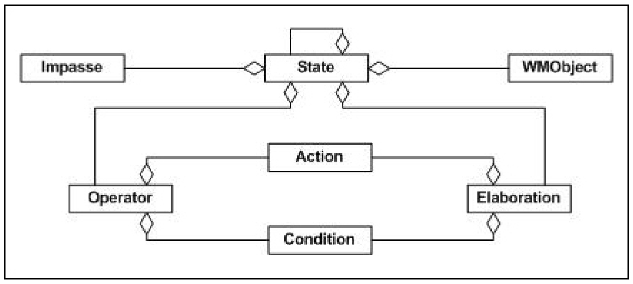

Figure 2.2 Herbal Ontology Class Relationships.

Herbal also makes it easier to reuse programming constructs both within the same model and across multiple models. For example, operators and elaborations can share conditions and actions, and states can share operators, elaborations, and impasses. Reusing components in this fashion reduces the amount of cutting and pasting that can be common when programming in Soar.

It is important to note that the use of the Herbal high-level language does not preclude the need to understand how to program in Soar. This language makes it easier to understand Soar models, generate code embedded with descriptive text, and promote reuse – but it does not make it possible to create Soar models without some basic understanding of Soar programming.

4. The Herbal Development Environment

The Herbal development environment (version 0.9 is available at acs.ist.psu.edu/projects/Herbal/) has two main responsibilities: to make it easy to graphically generate an Herbal high-level program, and to translate this program into Soar productions (Figure 4.1).

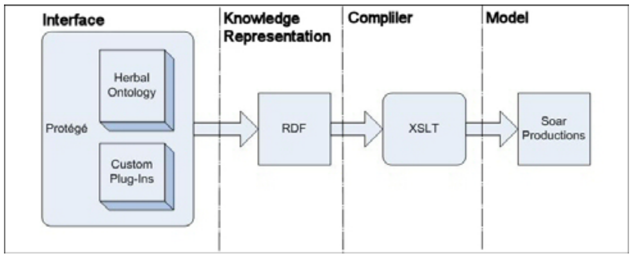

Figure 4.1 The Herbal Development Environment Software Architecture.

The first responsibility is realized using the Protégé integrated knowledge-based development tool (Stanford Medical Informatics, 2004). Without any customization, Protégé makes it possible to graphically define an ontology, create an instance of that ontology, and save it in Resource Description Framework (RDF) format (W3C, 2004c). By combining our own custom Protégé plug-ins, and the Herbal high-level language ontology, Protégé was easily configured to create a graphical programming environment for Soar models. Modelers use Protégé to instantiate a model by extending the Herbal ontology, and then saving this model in RDF format. Portions of existing models can be reused by importing existing model ontologies into Protégé and adapting as required.

The second responsibility of the Herbal development environment is compiling the generated RDF into a valid set of productions. The Extensible Stylesheet Language Transformations (XSLT) standard is used to compile the RDF into Soar productions (W3C, 2004a, 2004b).

Currently, Herbal supports the generation of Soar productions. However, our design makes it easy to support additional target architectures; to support a new cognitive architecture, a new XSLT script must be written and plugged into the Herbal development environment.

5. A Sample Model

To better illustrate how a model can be created using the Herbal development environment, a modified version of the blocks world model introduced in the Soar Manual and Documentation (Laird et al., 1999) was implemented using Herbal and is summarized here. Implementing this model is also described in The Herbal Tutorial (Cohen & Ritter, 2004).

This simple model consists of three blocks (A, B, and C) positioned on a table. The goal of the model is to stack these blocks such that block A is on top of block B, block B is on top of block C, and block C is on top of the table.

This model is articulated in the following sections by describing the different objects that were instantiated in the Herbal development environment. The Herbal blocks world model consists of working memory objects, state objects, operator/elaboration objects, impasse objects, and condition/action objects.

5.1 Working memory

The classes that represent the problem domain are created using Protégé, and referenced by other components of the model. For example, using the Herbal development environment the blocks world model defines three classes of working memory: Block, Table, and OnTop (the OnTop class represents a relationship where a block is on top of either a table or another block). Using these classes, three Block instances were created, one for each of the three blocks (A, B, and C). In addition, one Table instance was created, and three OnTop objects were created representing the fact that all three blocks start on top of the table. These classes and instances are illustrated in Figure 5.1.1.

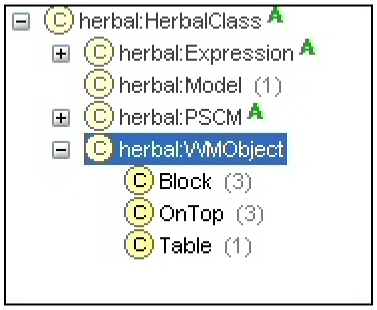

Figure 5.1.1 Blocks World Model Working Memory Objects.

5.2 States, operators and elaborations

The Herbal Blocks World Model also consists of two states: the top state (*BlocksWorldState*) and a sub state (*ResolveTie*).

The top state is adorned with the working memory objects shown in Section 5.1. In addition, the top state consists of an elaboration that checks to see if the goal has been reached, and two operators: *MoveBlockToTable* and *MoveBlockToBlock*. Finally, the top state has an operator-tie impasse object (*block-move-tie*) that is meant to handle the case where a choice must be made between moving a block to the table and moving a block on top of another block.

The *block-move-tie* impasse handles a tie between the *MoveBlockToTable* and *MoveBlockToBlock* operators using the *ResolveTie* state. The *ResolveTie* state contains two operators: *PreferAOnB* and *PreferBOnC*. These two operators designate a preference for one of the two tied operators using a strategy that places a priority on putting block B on block C, and then block A on block B.

A snapshot of the blocks world top state, taken from the Herbal development environment, is shown in Figure 5.2.1.

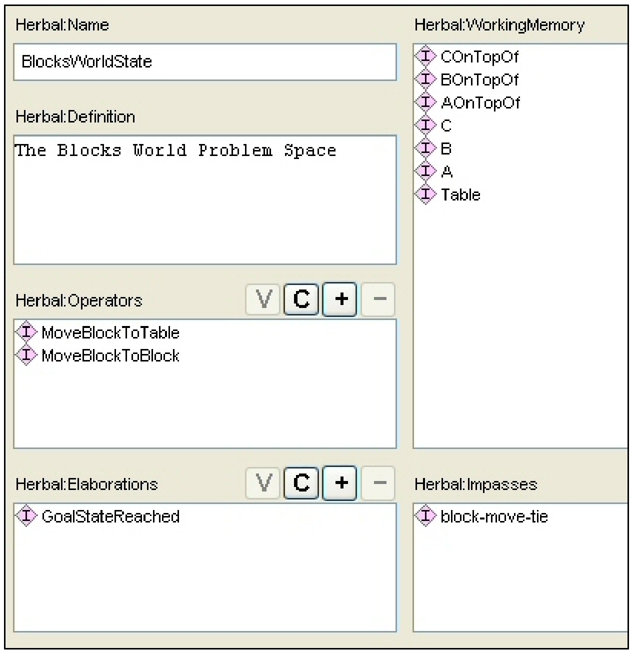
Herbal:Name
BlocksWorldState
Herbal:Definition
The Blocks World Problem Space
Herbal:Operators
MoveBlockToTable
MoveBlockToBlock
Herbal:Elaborations
GoalStateReached
Herbal:WorkingMemory
COnTopOf
BOnTopOf
AOnTopOf
C
B
A
Table
Herbal:Impasses
block-move-tie

Figure 5.2.1 Top State for the Blocks World Model.

Figure 5.2.1 illustrates how the theoretical components of a Soar model can be developed and browsed from within Herbal. For example, using the Herbal development environment it is easy to ascertain the fact that the top state in this model is made up of two operators, one elaboration, an impasse, and various working memory objects. Double clicking on any of the contained objects shown in Figure 5.2.1 (e.g. , the *MoveBlockToTable* operator) results in a dialog box that gives further detail on the selected object and its attributes. This information – typically implicit in Soar productions – is obvious from within the Herbal environment, which represents some of the most fundamental aspects of the explanation facilities included and planned for Herbal.

In addition to the components already mentioned, the blocks world model also consists of several conditions and actions. These objects are used as major building blocks for operators and elaborations and make it possible to share conditions and actions.

5.3 Compiling the model

Compiling the Blocks World Model is accomplished using the custom Protégé plug-in shown in Figure 5.3.1. The compiler also allows the user to include a header or other files in the compiled model. This code can, for example, initialize the model, hook up the model to agent environments like dTank (Morgan, Ritter, & Cohen, 2005), or include other models.

The generated Soar file for this model consists of over 300 lines of formatted and documented Soar code that contains 11 Soar productions.

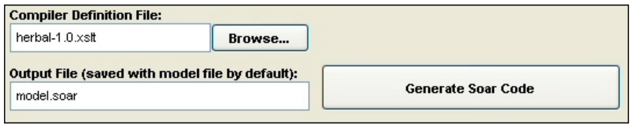
Compiler Definition File: herbal-1.0.xslt
Output File (saved with model file by default): model.soar

Figure 5.3.1 Herbal Protégé Compiler Plug-in.

The documentation included in the compiled productions is generated using information provided by the modeler when objects are instantiated in the Herbal language. All of the Herbal high-level language classes contain attributes such as “definition”, “purpose”, and “how-it-works”, information we found to be most important to explanation (Haynes et al., 2005), which are used by the compiler to generate useful comments. In addition, we are completing the design to use this same information at runtime to generate explanations.

6. Experiences Using Herbal in an Undergraduate Course

The Herbal development environment was used in IST 402 Emerging Technologies: Models of Human Behavior. This course was taught in the Fall of 2004, to 38 undergraduates and 3 graduate students, in the School of Information Sciences and Technology at The Pennsylvania State University at University Park. IST402 is a required course for IST majors, but each section covers a different emerging technology.

The course covers cognitive modeling, ontologies, and involves creating and testing working models. It used Soar, the Herbal development environment (version 0.7), and dTank (Morgan et al., 2005) as an example application domain.

Students provided positive feedback in their formal course evaluations on both Protégé and Herbal as modeling environments. In general, the students encountered fewer problems programming in Soar when using Herbal. This is admittedly an anecdotal result, and could also be the result of other factors. However, a recent single subject analysis of how long it takes to create a dTank model using the Herbal development environment (Morgan et al., in press) showed that Herbal can be as fast as TAQL (Yost, 1993)

A total of nine group final projects were created, and four out of nine used the Herbal development environment to create their final projects (all groups used Herbal in their homework). Two of the four teams that settled on Herbal had a model that ran, and one of the two that settled on using just Soar had a model that ran. The most complex of the working models was an Herbal model that contained 16 pages of rules with 25 operators. The students all took advantage of Herbal's facility to include a header in their code that hooked up the models to dTank as they loaded.

Among the advantages noted by students was Herbal's ability to reuse conditions and actions across several operators. This enabled teams to build more complex tanks using a set of predefined conditions and actions specific to the dTank environment. Students also commented on the usefulness of Herbal's ability to automatically generate comments.

At the time the course was taught, Herbal did not support impasses and this was noted by the students as one of the major disadvantages (impasses have since been added in version 0.9). Another disadvantage noted by the students was related to usability issues with Protégé. A new version of Protégé (version 3.0) has since been released that has addressed many of these issues.

7. Conclusion

We believe that the ability to create cognitive models using powerful architectures such as Soar can be simplified using high-level languages. The Herbal high-level language is an example of such a solution and is capable of producing Soar productions from models created using the Herbal development environment, which is based on a modified ontology editor.

In this paper, a simple model built using Herbal was described to help illustrate how Herbal simplifies programming in Soar. In addition, feedback provided by undergraduate students who used Herbal in a course at Penn State was discussed.

Future development of Herbal is currently focused on using the information contained in the Herbal ontology to generate explanations while the model is running. Runtime explanations should further aid developers, as well as other users, with the creation of cognitive models.

8. Acknowledgements

The development of this software was supported by the Office of Naval Research and was carried out under the terms of Contract No N000140210021. We also thank Geoffrey P. Morgan, Urmila Kukreja, Isaac Councill, and the students of IST 402 the last two years for their work with and comments on Herbal.

9. References

ACT-R Research Group Department of Psychology Carnegie Mellon University. (2004). *ACT-R 5.0 Tutorials*, from http://act-r.psy.cmu.edu/tutorials/

Busetta, P., Rönnquist, R., Hodgson, A., & Lucas, A. (1999). JACK Intelligent Agents - Components for Intelligent Agents in Java. *AgentLink Newsletter*.

Cohen, A. M., & Ritter, F. E. (2004). *The Herbal Tutorial* (No. Tech. Report No. 2004-2): Applied Cognitive Science Lab, School of Information Sciences and Technology, Penn State.

Haynes, S. R., Councill, I. G., & Ritter, F. E. (2004). Responsibility-driven explanation engineering for cognitive models. In R. M. Jones, R. E. Wray & M. Scheutz (Eds.), *AAAI Workshop on intelligent agent architectures: Combining the strengths of software engineering and cognitive systems* (pp. 46-52). Menlo Park, CA: AAAI Press.

Haynes, S. R., Ritter, F. E., Councill, I. G., & Cohen, M. A. (2005). Explaining Intelligent Agents. *Manuscript submitted for publication.*

Hirst, T. (1999). *ViSoar - Towards an Agent Development Environment for the Soar Architecture.* Paper presented at the 4th Online Workshop on Soft Computing (WSC4).

Laird, J. E. (1999). *Visual Soar.* Paper presented at the Soar Workshop 19, University of Michigan.

Laird, J. E., Congdon, C. B., & Coulter, K. J. (1999). *The Soar User's Manual Version 8.2*: University of Michigan.

Lebiere, C., Biefeld, E., Archer, R., Archer, S., Allender, L., & Kelley, T. (2002). *IMPRINT/ACT-R:*

Integration of a task network modeling architecture with a cognitive architecture and its application to human error modeling. Paper presented at the Advanced Technologies Simulation Conference, San Diego, CA.

Morgan, G. P., Cohen, A. M., Haynes, S. R., & Ritter, F. E. (in press). Increasing Efficiency of the Development of User Models. *IEEE System Information and Engineering Design Symposium*.

Morgan, G. P., Ritter, F. E., & Cohen, A. M. (2005). *dTank: An Environment for Architectural Comparisons of Competitive Agents.* Paper presented at the 14th Conference on Behavior Representation in Modeling and Simulation (BRIMS), Universal City, CA.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Ritter, F. E., Shadbolt, N. R., Elliman, D., Young, R., Gobet, F., & Baxter, G. D. (2003). *Techniques for modeling human and organization behavior in synthetic environments: A supplementary review*, Wright Patterson Air Force Base, OH: Human Systems Information Analysis Center.

Soar Technology Inc. (2004). *SoarDoc*, 2004, from www.eecs.umich.edu/~soar/sitemaker/projects/soardoc/soardoc.html

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.

Stanford Medical Informatics. (2004). Protege (Version 2.1.1).

Taylor, G. E., Jones, R. M., Goldstein, M., Frederiksen, R., & Wray, R. E. (2002). *VISTA: A Generic Toolkit for Visualizing Agent Behavior.* Paper presented at the Procedings of the Eleventh Conference on Computer Generated Forces and Behavioral Representation, Institute for Simulation and Training.

W3C. (2004a). The Extensible Markup Langauge.

W3C. (2004b). *The Extensible Stylesheet Language Family*, from www.w3.org/Style/XSL/

W3C. (2004c). *Resource Description Framework*, from www.w3.org/RDF/

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

Zachary, W., Jones, R. M., & Taylor, G. (2002). *How to communicate to users what is inside a cognitive model.* Paper presented at the 11th Computer Generated Forces Conference, Orlando, FL.

Author Biographies

MARK COHEN is an instructor in the Business Administration, CS and IT Department at Lock Haven University, and a PhD student 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.

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 published as a HSIAC State of the Art Report in 2003 (iac.dtic.mil/hsiac/S-docs/SOAR-Jun03.pdf).

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.