

Cohen, M. A., Ritter, F. E., Haynes, S. (2004). An introduction to Herbal. In Proceedings of the XXIV Workshop. 75-77. The Soar Group, University of Michigan.

An Introduction to Herbal

Mark A. Cohen (1,2) Frank E. Ritter (2,3) Steve Haynes (3)

1 BA CS & IT Department, Lock Haven University

2 Applied Cognitive Science Lab, 3 Penn State

Project Goals:

- Make it easier to learn how to program Soar
- Promote the reuse of Soar code
- Make it easier to understand running Soar models

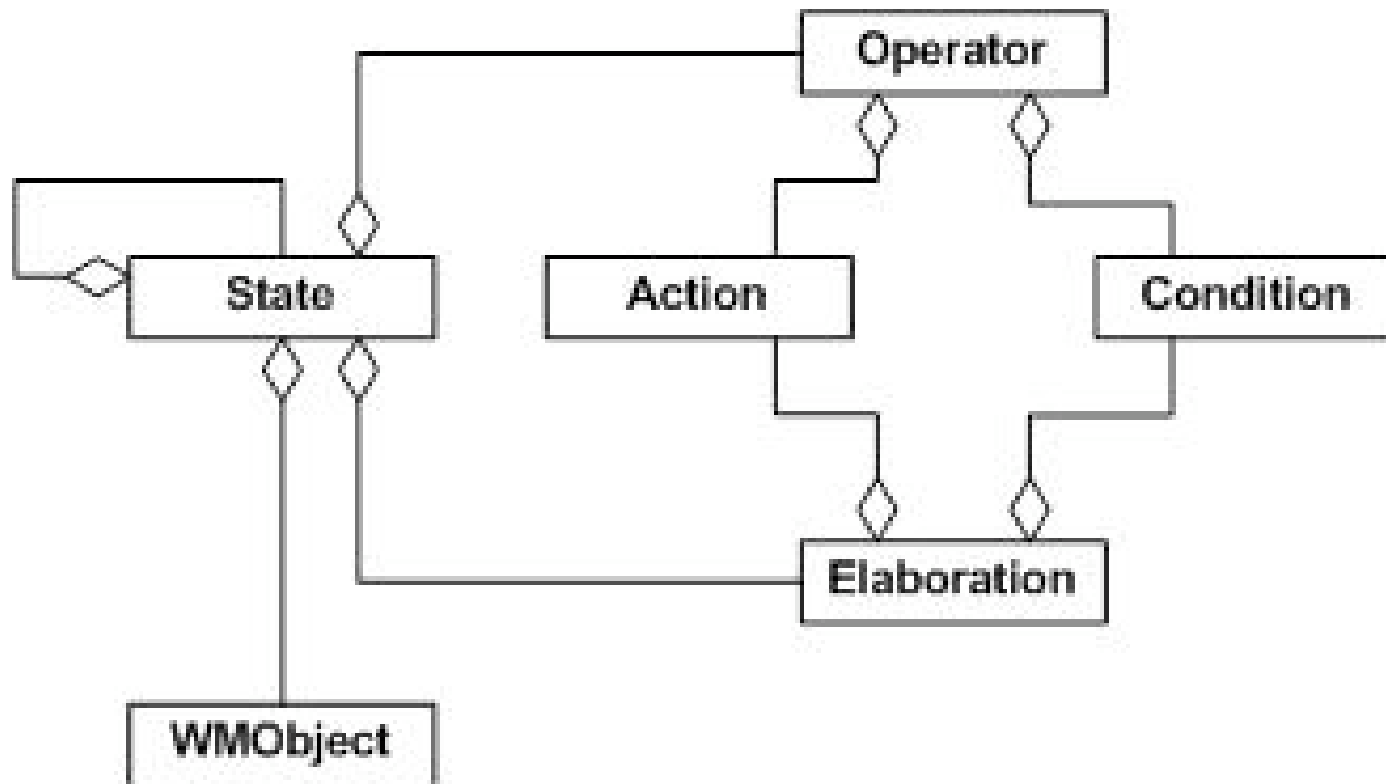
Tools that Support these Goals:

- Herbal High-Level Language
- Herbal IDE
- Herbal Viewer

Herbal High-Level Language

- The Herbal language is based on the definition of a Soar ontology
- The language is XML based (RDF) and can be compiled directly into Soar code using XSLT
- Developers can create Soar models in this language using a text editor, or with the help of the Herbal IDE

A Brief Look at the Herbal Soar Ontology



The Herbal IDE

- Used to simplify the creation of models using the Herbal High-Level Language
- Because models are written in Herbal by instantiating objects defined in an ontology, a graphical ontology editor seemed to be the best tool for the Herbal IDE -- thus Protégé was chosen for the Herbal IDE
- Protégé is a free ontology editor developed by Stanford Medical Informatics that makes it possible to:
 - Define a new ontology
 - Import an existing ontology for reuse
 - Create instances of classes in an ontology
- Protégé can be extended, as needed, via custom Java plug-ins

Creating A Simple Model Using the Herbal IDE

- Create a new project and import the Soar ontology
- Define and import the classes that make up your problem domain, and instantiate initial working memory
- Instantiate and import Soar objects that will define the behavior of your model
- Compile into running Soar code

Define and Instantiate the Problem Domain

The screenshot displays the Soar workspace interface, divided into three main sections:

- Classes:** A hierarchical tree view on the left showing the class structure. The root is `:THING`, followed by `:SYSTEM-CLASS`, `herbal:HerbalClass`, `herbal:Expression`, `herbal:Action` (3), `herbal:Condition` (3), `herbal:Operator` (3), `herbal:State`, `herbal:TopState` (1), and `herbal:WMObject`. Under `herbal:WMObject`, the `Block` class is selected, showing it has 3 instances, along with `OnTop` (3) and `Table` (1).
- Display Slot:** A central panel with a dropdown menu set to `herbal:Name`. Below the dropdown are several icons for editing and viewing the slot's content.
- Instance A:** A panel on the right showing the details of an instance of the `Block` class. The instance name is `A` (type=Block, name=blocksWorld-0.2_Instance_7). It contains three slots:
 - `Herbal:Name`: A text field containing the letter `A`.
 - `Herbal:Description`: A text field containing the text `A block with the letter A printed on it`.
 - `Clear`: A text field containing the value `true`.At the bottom, the `Type` slot is set to `Block`.

Defining Soar Objects

The screenshot displays the Soar Workshop interface with three main panels:

- Classes Panel:** A tree view of classes. The `herbal:TopState (1)` class is selected and highlighted in blue.
- Display Slot Panel:** Shows the selected class `BlocksWorldState` in a slot. Below the slot are icons for viewing, creating, deleting, and other actions.
- Object Definition Panel:** Shows the definition for the `BlocksWorldState` object.
 - Herbal:Name:** `BlocksWorldState`
 - Herbal:Description:** `The Blocks World Problem Space`
 - Herbal:Operators:** A list of operators: `MoveBlockToTable`, `GoalStateReached`, and `MoveBlockToBlock`.
 - Herbal:WorkingMemory:** A list of objects: `COnTopOf`, `BOnTopOf`, `AOnTopOf`, `C`, `Table`, `B`, and `A`.

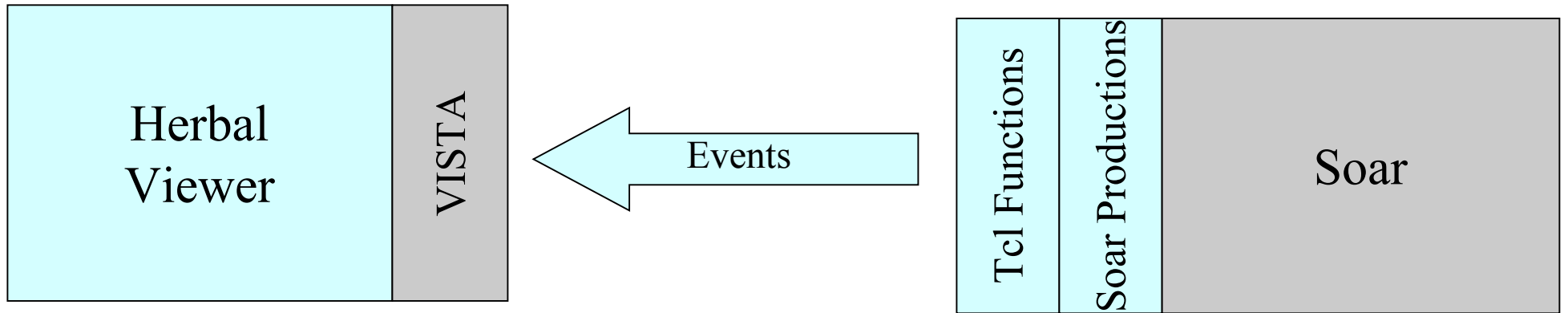
Defining Soar Objects (cont.)

The screenshot displays the Soar Workshop interface for defining a Soar object. On the left, a 'Classes' pane shows a hierarchical tree of classes, with 'herbal:Operator (3)' selected. The central 'Display Slot' pane shows a dropdown menu set to 'herbal:Name' and a list of objects including 'GoalStateReached', 'MoveBlockToBlock' (highlighted), and 'MoveBlockToTable'. The right pane shows the definition for 'MoveBlockToBlock' (type=herbal:Operator, name=blocks\World-0.2_Inst...). It includes fields for 'Herbal:Name' (MoveBlockToBlock) and 'Herbal:Preference' (Indifferent). The 'Herbal:Description' field contains the text 'Moves a clear block onto another clear block'. At the bottom, there are sections for 'Herbal:Actions' (containing 'MoveBlock1 OntoBlock2') and 'Herbal:Conditions' (containing 'TwoClearBlocks').

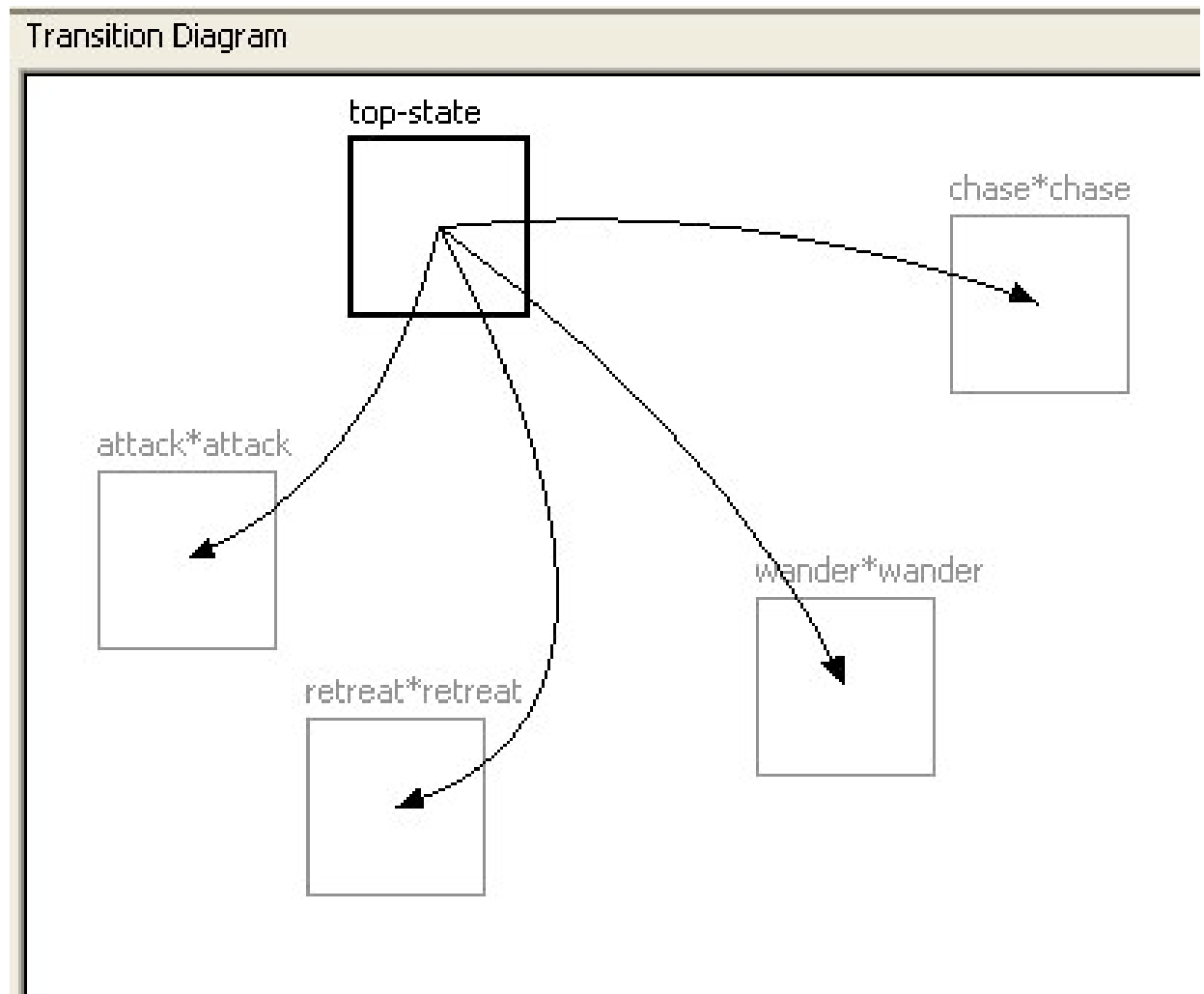
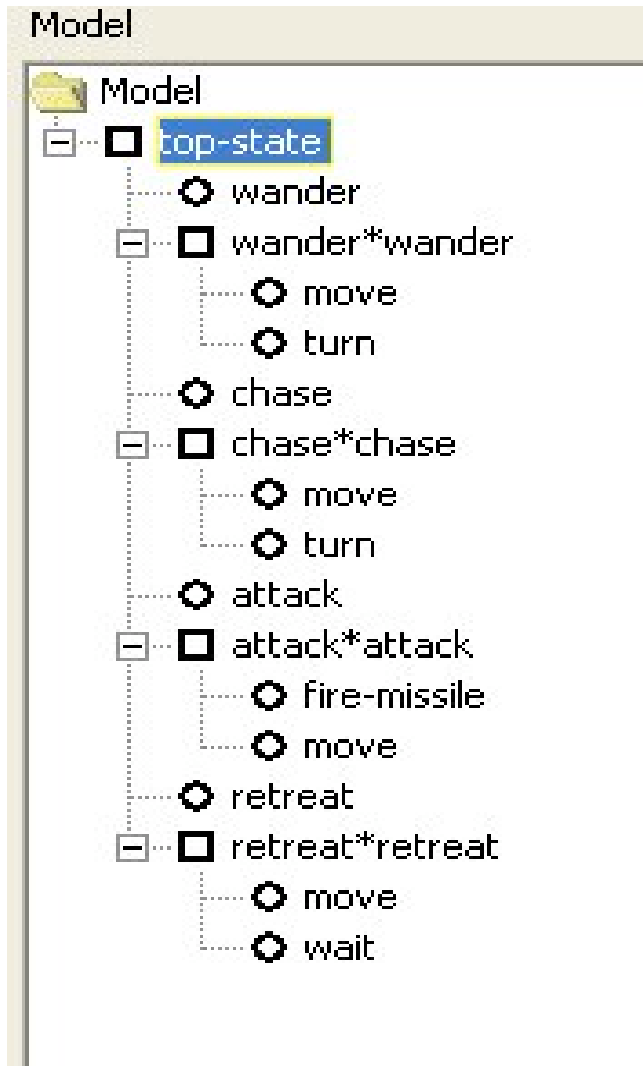
Herbal Viewer

- Generates displays of a running Soar model that will help explain the model's behavior
- Basic views work with all Soar models
- More advanced views and explanations are based on model structures from the Herbal High-Level Language

Viewer Architecture



The Tree and Graph View



Current Status/Challenges

- **Language**
 - We have created a simple blocks world model
 - When I return we will start building slightly more complex models: dTank, ToH
- **IDE**
 - Protégé is serving its purpose -- we will be developing a few plug-ins for customization
- **Viewer**
 - Currently works for Soar models but does not yet use the language for creating better displays with more explanation
- **All Three Tools**
 - Will be used at PSU this fall for IST 402 Models of behavior
- **Challenges/Lumps of Coal**
 - The Soar ontology is a moving target: we are trying to keep it simple and at the same time identifying ways to inject information that can be used for explanation by the viewer
 - Personnel changes

Acknowledgements/References

- Acknowledgements
 - Isaac Council
 - Kevin Tor
 - Geoff Morgan
 - Urmila Kukreja
 - Supported by Office of Naval Res. N00014-02-1-0021
- References
 - VISTA Developer's Handbook, Soar Technology Inc. (see Glenn Taylor)
 - Protégé: protégé.stanford.edu