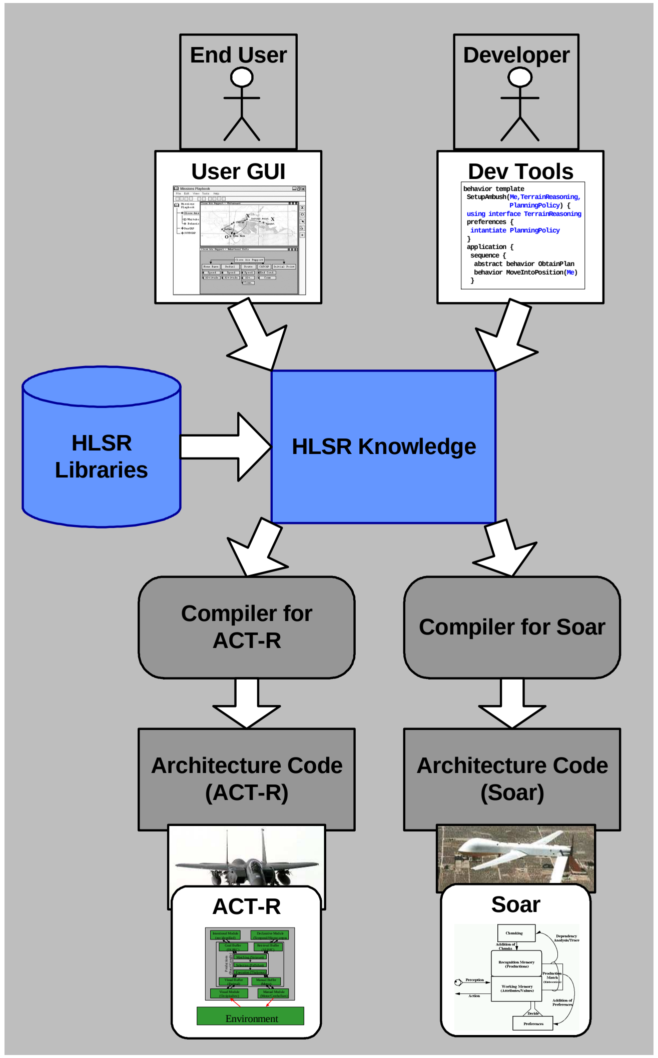
Improving HBM Affordability: A High-Level Language for Cognitive Architectures

Randolph M. Jones

Jacob Crossman, Christian Lebiere, Lisa Scott Holt,
Gil Barrett, David Ray, Kyle Aron, Nick Piegdon
Funded by the Office of Naval Research (ONR) Contract N00014-05-C-0245

What is HLSR?

- **H**igh **L**evel **S**ymbolic **R**epresentation
- A language for knowledge encoding
- The language is:
  - Architecture independent
  - Domain independent
  - High-level
  - Designed to support reuse
- Target users:
  - Behavior developers
  - End user tool developers



Primary accomplishments prior to 2007

- Review of cognitive/intelligent agent architectures
- Design of initial HLSR "virtual machine" and language
- Implementation of compiler parser and code generation
- Full implementation of code generation for Soar
- Partial implementation of code generation for ACT-R
- Initial comparative study of ACT-R, Soar, and HLSR programming
- *Internal Funding*: Developed an IDE and debugger for HLSR

Primary activities in 2007

- Finished implementation of code generation for ACT-R (for 2006 HLSR definition)
- Completed design and implementation of code-based metrics tools for HLSR, Soar, and ACT-R
- Designed I/O language elements in HLSR
- Designed and partially completed implementation of ACT-R and Soar code generation for new language elements
- Created TankHLSR models using the Tank Soar framework – DEMO will show this working
    - Evaluated these models using HLSR's metrics
- HLSR evaluation
    - Designed and executed user study to evaluate HLSR
    - Created algorithms for automated evaluation metrics
    - Improved understanding of HLSR strengths and weaknesses
    - Improved understanding of Soar and ACT-R architectural differences

Primary activities in 2008

- Final report for current ONR funding
- Outline and draft sections of potential journal paper

Comparative user study

- Goal: To evaluate potential advantages of HLSR over Soar and ACT-R for novice programmers
- Subjects:
  - Junior and senior computer science majors from Carnegie Mellon University plus a couple of graduate students
  - Cognitive modeling and AI programming experience not required
  - Volunteers randomly assigned to groups (ACT-R, Soar, HLSR)
- Design
  - 2 hour interactive tutorial (with exercises) in language followed by 1 hour exam
  - Each group learned only one language
  - ACT-R N=8; Soar N=6; HLSR N=9

Tutorials, exercises and exam problems

- Developed separate tutorials with interactive exercises for each language
  - All tutorials covered the same basic concepts
  - Exercise were the same across languages
- Kept exam problems simple
  - Subjects were novices
  - Not much time to learn or practice language before exam
- Designed exam problems to gauge:
  - Ability to understand existing code, including how it will behave dynamically (when executing)
  - Ability to make changes to existing code
  - Ability to design behavior using specific language constructs

Data Collection and Analysis

- Subjects self-recorded time spent on each exam problem (and sub-problem)
- Subject submitted all written work and solutions
- Experimenters coded various aspects of solutions
  - Quality, correctness, and format of program design
  - Quality and correctness of program code
  - Evidence of understanding of language concepts
  - Correct use of language-specific constructs
- Exams coded by language experts
  - ACT-R: Lebiere; Soar: Jones; HLSR: Crossman
  - Initial coding followed by group discussions and analysis for consistency

Hypotheses

- Major differences between groups were not expected
  - The problems were very simple
  - Hopefully some trends would still be visible
  - The problems were built from a Soar tutorial, and may have been biased toward "Soar-like thinking"
- Expectations
  - Tasks dealing with complex logic, sequences/loops, and declarative structures should be easier in HLSR (fewer mistakes, shorter time) **[not confirmed]**
  - Because they are at a higher level of abstraction, HLSR constructs should be used more often in design than ACT-R and Soar constructs **[positive trend, n too small for χ^2]**
  - Time taken to complete some tasks should be reduced in HLSR **[confirmed significant difference]**

HLSR I/O

- Why I/O over Other Language Features?
  - I/O is critical for almost any useful model
  - Practical: I/O can be implemented in the limited time remaining
- **Observation**: Both Soar and ACT-R
  - Treat I/O structure the same as declarative memory
  - Have I/O modules that run in parallel to decision cycle
- **Approach**: HLSR I/O leverages the relation – relations can be "sensed" (input) or "externalized" (output)
  - Conceptually input relations form an input pool
  - Input relations are "sensed" when the model sensors detect instance of what the relation represents
  - Output relations are visible to motor system
  - Output relations exist in declarative memory so output processes can be queried (meta reasoning over motor process)

Metrics – Results for Tank Model

	Metric	HLSR	ACT-R	Soar
Volume	LOC	134	780 (5.8x)	337 (2.5x)
	Tokens	516	1417 (2.75x)	892 (1.73x)
Encaps.	Objects per construct	2.9 objects/construct (construct = relation w. cond., transform, AT)	5.8 chunk types/goal 2.3 chunk types/rule	4.3 objs/operator 4.13 objs/production
	Attributes per construct	2.4 attr/construct	14.2 attributes/goal 4.8 attributes/rule	7.3 attr/operator 7.7 attr/production
Complexity	# Procedural Constructs	19 constructs 36 statements	6 goals 54 rules	9 operators 45 productions (36 are elaborations)
	# Tests	90 tests total 4.74 tests/construct	210 tests total 34.8 attr tests/goal 3.9 attr tests/rule	100 logical tests total 1.89 tests/operator 2.22 tests/production
	Average Fanning	4 fanning/act. table 3 fanning/transform 2.15 fanning/statement	1.67 fanning/goal 9.2 fanning/rule	0.67 fanning/operator 6.6 fanning/production

- Note we started with Soar model (optimal Soar model)

HLSR lessons learned in 2007

- HLSR's "activation table" construct appears to be quite powerful (especially in terms of saving lines of code), and was of particular interest in the user study
- HLSR's relation/goal/fact constructs withstand formalization and compilation in ACT-R and Soar, and work will in a variety of implemented models
  - Difficult implementation issues sometimes, but appears to be at a good level of abstraction
  - Provides a uniform construct that encompasses various ACT-R and Soar modeling patterns (Soar i-support, ACT-R retrieve-best, goal/belief maintenance)
- Demonstrated 2-3x code reduction in small problem domains
  - Reductions increased with move to more complex problems (using I/O)
  - More reduction should be possible with language features that have been partially designed but not implemented
- User study was not conclusive in all respects, but
  - HLSR subjects spent less time on some problems (with comparable correctness) than ACT-R and Soar subjects
  - Differences between design and code appear to be smaller for HLSR
- There are interesting low-level modeling differences that Soar and ACT-R languages constrain modelers to use.
  - HLSR provides a method for formalizing these differences and encouraging consistent modeling solutions.
  - Should allow improved consistency and comparison of models within and across architectures.

HLSR issues identified in 2007

- The transform language construct is too limited
  - Does not cover some of the more flexible modeling patterns in Soar and ACT-R
  - Does not really make difficult procedures much easier to write
  - RESPONSE: Divide transform into two separate constructs. One for simple sequences of actions, others to support conditions, looping, parallelism, and aspect-oriented programming
- The relation language construct should have slightly different semantics
  - Current implementation makes asserted facts immutable (ala ACT-R), which leads to programming at an unnecessary level of detail
  - Needs an inheritance system for richer declarative knowledge specifications
- Design and implementation of parallelism needs to be enhanced and improved