Intelligent Behaviors for Simulated Entities

Dr. Dan Fu, Ryan Houlette, Jeremy Ludwig
Stottler Henke Associates
San Mateo, CA
fu@stottlerhenke.com, houlette@stottlerhenke.com, ludwig@stottlerhenke.com

TUTORIAL ABSTRACT

In nearly any simulation system, there will be entities – that is, platforms or forces – that are not under the control of a human participant, either because the necessary personnel are not available or they would be too costly. These entities must mimic the behavior of the real-world entities that they represent in order to achieve some level of believability in the simulation. Crafting realistic, intelligent-seeming behaviors for simulated entities is a non-trivial task, however, and there are a variety of techniques that can be employed. While there exists a diverse array of agent architectures, two common approaches are cognitive architectures and state machines.

This tutorial provides an introduction to creating intelligent behaviors for simulated entities. An overview of the most common approaches will be given, and the advantages and disadvantages of each will be discussed. Examples of actual implemented behaviors will be used to illustrate important concepts. The tutorial will also present a general process for authoring behaviors, including typical steps and common pitfalls.

Attendees of this tutorial will come away with an overview of the various options available for adding intelligent behaviors to simulated entities. It will also provide them with the information they need to choose the appropriate approach for their own applications, along with guidelines on good authoring practices.

ABOUT THE PRESENTERS

Dan Fu joined Stottler Henke six years ago and has worked on several artificial intelligence (AI) systems including AI authoring tools, wargaming toolsets, immersive training systems, and AI for simulations. Dan was principal investigator on an intelligent agents project to create AI middleware for simulations and videogames. The result was SimBionic, which enables users to graphically author entity behavior for a simulation or videogame. Dan holds a B.S. from Cornell Univ. and a Ph.D. from the Univ. of Chicago, both in computer science.

Ryan Houlette holds a Master's Degree in Computer Science with a concentration in Artificial Intelligence from Stanford University and a Bachelor of Arts Degree in Computer Science from DePauw University. He has been with Stottler Henke since 1998. He is the lead architect of the SimBionic product line, which features a behavior engine and graphical authoring tool allowing non-programmers to specify the behavior of virtual entities within military training simulations. He also recently led the development of a mixed-initiative intelligent interface framework that includes as a core component a rich capacity for human interaction and collaboration. Mr. Houlette has been involved in the design and implementation of a variety of other AI systems at Stottler Henke as well.

Jeremy Ludwig joined Stottler Henke in the fall of 2000 after completing his Master's Degree in Computer Science, with a concentration in Intelligent Systems, at the University of Pittsburgh. Recently, Mr. Ludwig has been a part of the SimBionic project team. He is also the lead developer on a Phase III simulation and training project, deployed at NAS North Island, for the Navy's common cockpit helicopter. Other research involves building cognitive models with the ACT-R, EPIC, and Soar cognitive architectures. Completed projects include developing an ontology, including temporal constraints, for a commercial project using JESS and designing methods for using individual differences to improve student learning in an intelligent tutoring system for the Army's initial entry rotary wing training.

Intelligent Behaviors for Simulated Entities

Dr. Dan Fu, Ryan Houlette, Jeremy Ludwig
Stottler Henke Associates
San Mateo, CA
fu@stottlerhenke.com, houlette@stottlerhenke.com, ludwig@stottlerhenke.com

TUTORIAL AIM

In nearly any simulation system, there will be entities – that is, platforms or forces – that are not under the control of a human participant, either because the necessary personnel are not available or they would be too costly. These entities must mimic the behavior of the real-world entities that they represent in order to achieve some level of believability in the simulation. Crafting realistic, intelligent-seeming behaviors for simulated entities is a non-trivial task, however, and there are a variety of techniques that can be employed. While there exists a diverse array of agent architectures, two common approaches are cognitive architectures and state machines.

This tutorial provides an introduction to creating intelligent behaviors for simulated entities. An overview of the most common approaches will be given, and the advantages and disadvantages of each will be discussed. Examples of actual implemented behaviors will be used to illustrate important concepts. The tutorial will also present a general process for authoring behaviors, including typical steps and common pitfalls.

Attendees of this tutorial will come away with an overview of the various options available for adding intelligent behaviors to simulated entities. It will also provide them with the information they need to choose the appropriate approach for their own applications, along with guidelines on good authoring practices.

LEARNING OBJECTIVES

- Gain awareness of the range of available techniques for developing "intelligent" simulated entities.
- Understand advantages and disadvantages of each approach.
- Understand how to implement simple behaviors using several common approaches.
- Learn a general process for authoring intelligent behaviors.

BACKGROUND

A large number of agent architectures have been developed to represent human behavior in simulated entities, especially for entities situated within military simulations (Pew & Mavor, 1998). Given that a comprehensive review of all (or even many) of the existing architectures is beyond the scope of a tutorial, this tutorial will instead concentrate on two commonly used methodologies in behavior modeling: cognitive architectures and state machines. These two different paradigms tend to represent the extreme viewpoints for a variety of modeling dimensions as will be discussed later.

Byrne writes that "a cognitive architecture is a broad theory of human cognition based on a wide selection of human experimental data and implemented as a running computer simulation program." Lehman, Laird, and Rosenbloom (1998) provide a slightly different definition: "a cognitive architecture is really two things at once. First, it is a fixed set of mechanisms and structures that process content to produce behavior. At the same time, however, it is a theory, or point of view, about what cognitive behaviors have in common."

Both of these definitions are based on well defined psychological theory of cognition, such as the Model Human Processor (Card, Moran, & Newell, 1983) . The theory behind a cognitive architecture describes the various processes involved in cognition, how these processes are related to and communicate with one another, and the types of information that these processes work with.

For this tutorial two different cognitive architectures will be examined, each based on a slightly different theory: ACT-R (Anderson et al., 2004) and Soar (Laird & Congden, 2004). Both of these architectures are extensively validated, commonly used, and freely distributed.

Formally, a finite-state machine (FSM) starts in its simplest form as a construct from computational theory, defined as a set of states S, an input vocabulary I, and a transition function T(s,i) mapping a state and an input to another state. The machine designates a single initial state designated as the *start* state, and zero or more *accepting* states. After the FSM processes all input, the ending state's identification as an accepting state dictates whether the machine accepts the input or not.

Less abstractly, an FSM is a concise, non-linear description of how an object can change its state over time, possibly in response to events in its environment. Its practical use departs from the theoretical definition in four ways. First, because it's intuitive to think of each state as representing some desired behavior, each state has corresponding code so that as the object's state changes, its behavior changes accordingly. Second, the monolithic transition function T resides across states. Each state can be said to "know" the conditions under which it should transition to a different state. Third, the notion of an accepting state is irrelevant. In its stead, accepting states are generally interpreted as the end of execution for the FSM. From here, another FSM gets invoked by the game to handle further input. Fourth, the input continues indefinitely until the FSM is no longer needed, or the game ends.

FSMs are a fairly common method of representing behavior for a wide variety of domains. Because of this, there exist a relatively large number of FSM packages. Because we have expertise in this area, we selected the SimBionic system for use in this tutorial. (Fu, Houlette, Jensen, & Bascara, 2003) It is an example of a mature FSM tool designed specifically for modeling intelligent behavior.

GENERAL METHODOLOGY FOR AUTHORING INTELLIGENT BEHAVIORS

While this tutorial concentrates on two particular types of modeling, there are a number of underlying similarities that apply to most modeling efforts. The first is what is often referred to as the *Cognition-Artifact-Task* triad (Gray & Altmann, 1999). The capabilities of a model arise from the interaction of these three distinct components. *Cognition* is the abilities and constraints supplied by the agent architecture. *Artifact* is what the model interacts with. In this tutorial, the artifact is generally assumed to be a simulation of some sort. *Task* is the specific knowledge needed to complete a task, such as controlling a bot in a first-person shooter. This knowledge is written in the language of the architecture and brought to bear at run-time when simulating behavior for an entity. A model is an interactive system that combines all three of these elements. In building a model, the general steps are to (1) identify/build a simulator, (2) integrate the agent architecture with the simulator, (3) write the task knowledge in the language of the architecture, (4) run the architecture to generate agent behavior and compare with desired results, and (5) adjust the task knowledge and return to (4).

Another, related, similarity is the difficulty of gathering and verifying task knowledge. It is important to consider that acquiring and codifying task knowledge requires knowledge-engineering and model-programming skills to specify the task, specify the task environment, and create a set of instructions for a particular agent architecture that completes the task. For instance, One of the interesting findings of TacAir-Soar (Jones et al., 1999) was that roughly 70-90% of the model development time was spent in the communication process – moving information from the subject matter expert to the programmer (Pearson & Laird, 2004). This includes both time spent in pilot model creation and pilot model verification for a number of different mission types. In both cases, the subject matter expert has the knowledge of what the model should look like or should be doing but the developer needs to create/modify the behavior. This problem has spurred on a number of research projects, including modification and verification of existing large rule sets (Pearson & Laird, 2004) and automatically comparing expert and model behaviors (Wallace & Laird, 2003).

COGNITIVE ARCHITECTURES

Specific Learning Objectives

- Appreciate some of the features, similarities, and differences of the Soar and ACT-R architectures.
- Understand the basic methodology for creating models in both architectures.
- Gain awareness of some of the modern applications of these two architectures.

Summary of Material

A simple definition for a cognitive architecture is: a psychological theory realized as a cognitive model programming language and runtime interpreter. The programming language dictates the representation of knowledge and provides a number of basic functional operators used in model construction. The runtime interpreter executes cognitive models according to the underlying theory, supplying input from the outside

world (perception), performing perceptual, cognitive, or motor processing as described by the model and constrained by the architecture (cognition), and producing actions (behavior).

For example, here are a few ACT-R operators:

- Check the current contents of a buffer (visual, goal, long term memory)
- Perform a visual search, or long term memory search, for an item that matches a description and insert it into the appropriate buffer
- Change the contents of the current goal
- Press a key on a simulated keyboard
- Add a *chunk* to long term memory

And some for Soar:

- Propose an operation to perform
- Apply preferences to different available operations
- Perform a selected operation
- Elaborate the current state

These cognitive architecture operators provide predictive power to cognitive models in part by constraining what a model can do during its execution (Kieras, 2003). These constraints help provide psychological validity for a given model as the operators are carried out within the runtime portion of the architecture. The goal is to allow people that aren't cognitive psychologists to construct psychologically valid models using the language of a cognitive architecture.

In order to make these architectures more concrete, a brief summary of the architectures is given, followed by the discussion of a model written in each architecture.

ACT-R

A much more detailed description of the architecture than what follows can be found in Anderson et al. (2004). ACT-R is composed of a set serial modules running in parallel on a 50 ms clock cycle: perceptual, motor, goal, declarative memory, and cognitive (procedural memory). The cognitive module communicates with the other systems through independent buffers, each of which holds only "chunk" at a time (Anderson et al., 2004).

The cognitive module uses the chunk information in the buffers for task execution. In ACT-R task knowledge is represented as production rules of the type IF the buffers contain certain information THEN perform some operation. During any given cycle, multiple productions may be able to fire. However, only one of these productions can fire in a given cycle. A production is chosen to fire based on an *expected utility* equation.

ACT-R also features a symbolic method for procedural learning called production compilation. This feature attempts to combine any two productions that occur in sequence into a single production with the effect of both productions. These new productions improve performance in two ways. First, one production is more efficient than two productions. Second, new productions can directly encode previously retrieved declarative memory thus reducing the number of declarative memory retrievals in the future.

More information on ACT-R, including software, publications, and models can be found at: http://act-r.psy.cmu.edu/

ACT-R Models

A verb-tense usage model from the ACT-R tutorial is examined to illustrate the architecture (*Unit 7: Production Rule Learning*). Here is the description of how the model should work given in the tutorial:

> "The learning process of the English past tense is characterized by the so-called U-shaped learning in the learning of irregular verbs. That is, at a certain age children inflect irregular verbs like "to break" correctly, so they say "broke" if they want to use the past tense. But at a later age, they overgeneralize, and start saying "breaked". At an even later stage they again inflect irregular verbs correctly."

This model contains some production rules that control verb tense formation and learns others as the model execution proceeds. The utilities of both kinds of production rules, which control which production rules are fired, need to be learned from "hearing" verbs from the environment. The initial rule set for this model, as well as learned rules, will be analyzed in the tutorial along with the ACT-R model development environment.

While this particular model is a bit academic, other ACT-R models are more relevant to military simulations. Some of these include the ACT-R/IMPRINT hybrid pilot model(Craig et al., 2002) and software that converts graphical depictions of behavior to ACT-R models run by Unreal Tournament bots (Douglass, 2003).

Soar

The definitive description of the Soar architecture is provided by Laird & Congden (2004). In their paper they state that the basic component of the Soar architecture is the problem space. To achieve *goals,* Soar uses *operators* to move through a *problem space* represented by *states* that consist of attributes with values and contain a goal and possibly parent and/or child states. The states, with parents and children, form a goal hierarchy. Long term memory (LTM) is made up of productions. It represents general knowledge (such as knowledge about things in general). Working memory (WM) contains the current state (such as knowledge about a particular thing), as well as the state hierarchy.

The *decision cycle* applies LTM to the current state. There are three stages in the decision cycle: propose operators, select operator, and apply operator. In the propose operator phase, all *elaborations*, operator propositions, and operator comparisons are fired in parallel. This phase continues until no more productions apply (quiescence). Then, from the proposed operators, one is selected.

If an operator cannot be selected an *impasse* is reached. To resolve this impasse, a new *substate* is created in which Soar attempts to resolve the impasse. This new state is a copy of the current state, but with a goal of resolving the impasse. If resolved, Soar's *chunking* mechanism creates a new LTM production to remember what to do if this impasse arises again. This new production contains the relevant features of the state prior to the impasse with the relevant action (e.g., operator comparison; operator proposal).

More information on Soar, including software, publications, and models can be found at: http://sitemaker.umich.edu/soar

Soar Models

TankSoar is a simple game included in the Soar distribution. Working through the Soar tutorial chapters, several different models are made for controlling the behavior of a tank (Laird, 2004). One of the more complex TankSoar models will be examined in this tutorial as an example of developing a Soar model. Additionally, the modeling environment and recent improvements to Soar/Simulator integration technology will be discussed in the tutorial.

With respect to other, less pedantic, Soar models, one often cited success story of cognitive modeling is TacAir-Soar (Jones et al., 1999). It is an important model for a number of reasons. Foremost among these are the model's size (over 5200 productions, 400 operators, 100 goals) and successes (types of planes, number of missions, evaluated performance).

Questions

- What are the basic steps involved in creating a model with ACT-R? With Soar?
- Where are production rules found in ACT-R? In Soar?
- What are some of the basic operators used in these production rules?
- What is the basic representation mechanism for declarative facts in ACT-R? In Soar?
- Give an example of an ACT-R model and a Soar model.

STATE MACHINES

Specific Learning Objectives

- Appreciate the tradeoff between models developed in state machines and models developed in cognitive architectures.
- Understand the basic methodology for creating models in SimBionic.
- Be able to construct a simple FSM behavior.

Summary of Material

Various forms of FSMs, an artificial intelligence based modeling alternative to cognitive architectures, are often used for developing intelligent agents in the game industry (Fu & Houlette, 2003). FSM modeling is especially useful when game character behavior can be modeled as a sequence of different character "mental-states".

Some advantages that hierarchical FSM models (a FSM that can transfer control to other FSMs) possess over cognitive models are that they are generally easier to write, maintain, and verify, as well as being more efficient at runtime, than cognitive models. There are also a number of drawbacks to using FSMs. Since they are not psychologically validated, and generally provide no psychological constraints, their behavior often does not accurately model human behavior at small grain sizes or during human-agent interaction. They also tend to lack some of the interesting capabilities associated with cognitive architectures, such as planning, problem solving, models of perception and attention, and learning. Based on these advantages/disadvantages, FSM models tend to be on

the opposite end of the spectrum from cognitive models. FSM models are easier to write and faster to run, but lack the representational power of cognitive architectures.

To make the idea of an FSM model more concrete, the SimBionic architecture will first be examined, followed by an example model.

SimBionic

The core of SimBionic is a visual authoring tool where users specify situation assessment, analysis, and decision-making algorithms by drawing flow chart-like diagrams that specify sequences of conditions and actions. That is, in SimBionic the behavior of an entity is directed by a type of flow chart. The modeling language also contains advanced features for cooperating intelligent entities that execute hierarchical, polymorphic behaviors and share information using messages and blackboards.

SimBionic behaviors are subprograms that dynamically determine how each entity selects actions to perform. Each behavior is comprised of:

- *Actions* carried out by the entity in the application, such as Fire weapon.
- Other *behaviors* which can be invoked by the current behavior.
- *Conditions* which check whether the application is in a specified state. Conditions can invoke predicates that check the state of the application, such as 'Is there a threat nearby?'
- *Connectors* which connect actions, behaviors, and conditions to specify their relative order of execution.

It is fairly simple to integrate SimBionic with an arbitrary simulator. The simulator, or a wrapper around the simulator API needs to perform a number of tasks. The first is instantiating a SimBionic runtime engine. This involves completing, in the programming language, methods that perform the *conditions* and *actions* used in the behaviors. The second step is to call *update* on the SimBionic engine to advance the behaviors.

SimBionic Models

For SimBionic, a model that controls terrorist and counter-terrorist bots in a first person shooter (Counter-Strike) will be examined in the tutorial. This model will illustrate the major SimBionic features as well as discuss the SimBionic development environment.

Questions

- What are some of the tradeoffs between modeling in SimBionic vs. ACT-R or Soar?
- What is the basic methodology for constructing a SimBionic model?
- Develop a simple behavior using a flow-chart like drawing.

CONCLUSION

This tutorial describes human behavior modeling in three specific environments selected from two quite different modeling areas. In the course of this discussion, a number of general issues applicable to modeling in a variety of agent architectures are encountered.

ACKNOWLEDGEMENTS

This work was supported in part by Air Force Research Laboratory grants F30602-00-C-0036 and FA8750-04-C-0085. Special thanks to David Ross of AFRL/IFSB, david.ross@rl.af.mil, the AF technical representative.

REFERENCES

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of mind. *Psychological Review, 111*(4), 1036-1060.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, N.J.: L. Erlbaum Associates.

Craig, K., Doyal, J., Brett, B., Lebiere, C., Biefeld, E., & Martin, E. A. (2002). *Development of a hybrid model of tractical figher pilot behavior using IMPRINT task network modeling and the adatpive control of thought - rational (ACT-R).* Paper presented at the 11th Conference on Computer Generate Forces and Behavior Representation.

Douglass. (2003). *Modeling of Cognitive Agents*. Retrieved May 22, 2005, from http://act-r.psy.cmu.edu/~douglass/Douglass/Agents/15-396.html).

Fu, D., & Houlette, R. (2003). The ultimate guide to FSMs in games. In S. Rabin (Ed.), *AI Game Programming Wisdom 2*.

Fu, D., Houlette, R., Jensen, R., & Bascara, O. (2003). *A Visual, Object-Oriented Approach to*

Simulation Behavior Authoring. Paper presented at the Industry/Interservice, Training, Simulation & Education Conference.

Gray, W. D., & Altmann, E. M. (1999). Cognitive modeling and human-computer interaction. In W. Karwowski (Ed.), *International Encyclopedia of Ergonomics and Human Factors* (pp. 387-391). New York: Taylor & Francis, Ltd.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., & Koss, F. V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine, 20*(1), 27-41.

Kieras, D. E. (2003). Model-based evaluation. In J. A. Jacko & A. Sears (Eds.), *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications* (pp. 1139-1151). Mahway, NJ: Lawrence Erlbaum Associates.

Laird, J. E. (2004). *The Soar 8 Tutorial*. Retrieved August 16, 2004, from http://sitemaker.umich.edu/soar/soar_software_downloads

Laird, J. E., & Congden, C. B. (2004). *The Soar User's Manual Version 8.5 Edition 1*. Retrieved August 16, 2004, from http://sitemaker.umich.edu/soar/soar_software_downloads

Lehman, J. F., Laird, J. E., & Rosenbloom, P. S. (1998). A gentle introduction to SOAR: An architecture for human cognition. In D. Scarborough & S. Sternberg (Eds.), (2 ed., Vol. 4, pp. 212-249). Cambridge, MA: MIT Press.

Pearson, D., & Laird, J. E. (2004). *Redux: Example-driven diagrammatic tools for rapid knowledge acquisition.* Paper presented at the Behavior Representation in Modeling and Simulation, Washington, D.C.

Pew, R. W., & Mavor, A. S. (1998). *Modeling human and organizational behavior : application to military simulations*. Washington, D.C.: National Academy Press.

Unit 7: Production Rule Learning. Retrieved October 31, 2004, from http://act-r.psy.cmu.edu/tutorials/unit7.htm

Wallace, S. A., & Laird, J. E. (2003). *Comparing agents and humans using behavioral bounding.* Paper presented at the International Joint Conference on Artificial Intelligence.