Creating Complex Actors with EASE

Paul Scerri
Dept. of Computer and Information Science
Linköping University, SE-58183 Linköping
pausc@ida.liu.se

Nancy E. Reed
Dept. of Computer and Information Science
Linköping University, SE-58183 Linköping
nanre@ida.liu.se

ABSTRACT

This paper presents a system called EASE (End-user Actor Specification Environment) that provides tools and methods to support end-user development of intelligent actors. The tools support the whole development process from design to testing. An EASE actor is a multi-agent system where a process of contract making and negotiation between agents determines the actions of the actor.

1. OVERVIEW OF EASE

Drawing on previous work [4, 5, 7] this paper presents a system that is intended to reduce the effort involved in actor development. EASE is a group of tools and an associated methodology for the development of complex, intelligent actors. In particular the system is intended to represent the first step toward putting actor development capabilities into the hands of the domain experts. By providing structure and support for a simple, rapid development process along with an accessible actor architecture, EASE provides the basis for end-user development of actors.

Within EASE an actor specification consists of a hierarchy of *agents* where each agent is responsible for some aspect of the overall actor's behavior. Each agent tries to accomplish only it's specific task and is hence fairly simple. Agents at lower levels in the hierarchy perform parts of behaviors for agents above them. At runtime an actor's specification is turned into a multi-agent system where overall actor behavior is determined by a continuous process of contract making and negotiation between agents. Agents at the bottom of the hierarchy negotiate amongst themselves over the actual output of the actor.

On top of the multi-agent actor architecture EASE enforces a methodology for actor development that covers all stages of development, from design through to reuse. The tools have been designed to make development easy by providing a completely graphical environment and reuse easy by enforcing strict modularity. Integrated tool support exists for quickly inspecting and debugging actors at runtime. The development aids in EASE combined with an underlying powerful agent runtime engine allow relatively inexperienced users to create useful actors for complex simulation environments.

The initial target domain for this system is simulated aircraft pilots. The TACSI air-combat simulator [1], developed at Saab AB, is used for both training of human pilots and testing of new systems. In this domain new actor behavior is often required and it is desirable that the engineers and pilot trainers that actually use the simulator can define the behavior. The pilot actors need to appear to be intelligent and act realistically in a very complex environment. The actuators for the actor, i.e. the aircraft controls, are extremely complex and allow many degrees of freedom.

2. ACTOR DEVELOPMENT

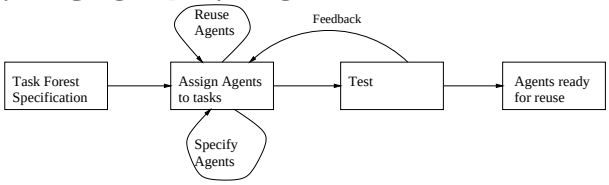

Figure 1: The actor development process

EASE supports the entire development process, as shown in Figure 1. Many of the development stages are explicitly supported by tools while for others a methodology is advocated. The development of an actor begins with the design of tree(s) of tasks and behaviors that describe the overall behavior of the actor. The design need not consist of a single tree but may be a forest of trees. The tops of trees represent high level abstract behaviors of the actor. Further down the tree are more specific aspects of the actor's overall behavior.

The next stage of development is to assign an agent to each node in the task forest. The agent will be performing the task assigned to that node. Leaf nodes are called "engineers" and internal nodes "managers". Branches between nodes are replaced by contract specifications. The internal behavior of the agents enforces the sequencing of tasks. When appropriate agents cannot be found in existing libraries they are created in the core of the EASE system, the agent specification tool shown in Figure 2). To specify an agent the designer specifies a name, intrinsic priority, environmental priority

To appear in the Forth International Conference on Autonomous Agents, June, 2000, Barcelona, Spain.

function, a state machine for controlling behavior and any contracts or factory assignments the agent will have.

The state machine for an agent provides the mechanism by which the decision making of an individual agent is defined. State transition conditions are defined with a function specification system. Engineer agents negotiate with a specific factory (i.e. over a particular degree of freedom) for each state in the state machine. In contrast, the designer specifies the contracts that manager agents should make in each state. Specifying a contract consists of selecting which agent should be contracted and instantiating any parameters associated with the contracted agent (e.g. the waypoint to fly toward). Once an actor has been partially or fully specified it can be tested in the target simulation environment. In order to support an iterative process of testing and incrementally expanding or improving actor behavior a number of graphical interfaces display interactively and in real-time the status of the agent's reasoning system.

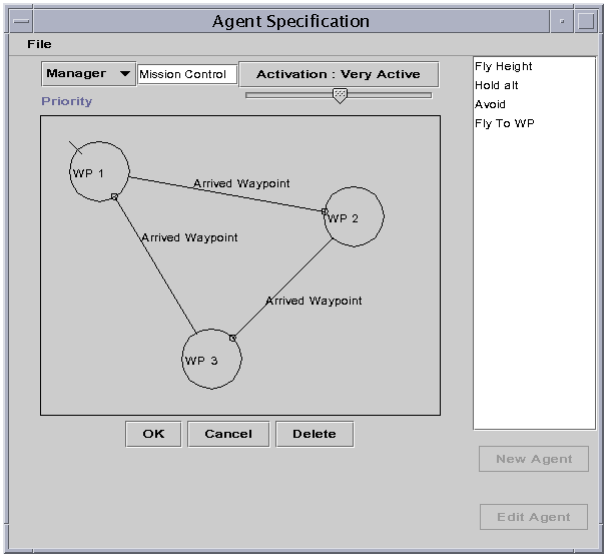

Figure 2: The Agent Specification subsystem

3. DISCUSSION

Rather than complex, monolithic systems incapable of interacting with the real world, Brooks [2] advocates a behavior-based approach where the overall behavior of a situated actor is broken *horizontally* into smaller pieces of behavior. An intuitive breakdown of overall actor behavior and a low risk incremental development process seem possible. For users unaccustomed to building actors, this paradigm potentially offers a specification that matches an intuitive breakdown of the actors' task. One of the difficulties in creating these systems has been that subtle interactions between individual behaviors means that the complexity of adding new behaviors to an actor soon becomes overwhelming [3].

Recently a variety of different approaches have been taken to reduce the complexity of the interactions or, at least provide methods for allowing developers to handle the complexity better. The methods either organize behaviors differently, e.g. [6]or combine the outputs of the behaviors in a different way, e.g. [8].With the EASE actor architecture we are taking this trend one step further.

A behavior in a behavior-based system is an agent in EASE. By elevating behaviors to the status of agents the interactions between behaviors simplify in the same way that encapsulating objects in other complex systems simplifies the interactions between subsystems. In effect, using agents instead of behaviors makes behaviors "active" rather than passive entities. The interaction between agents can then be strictly controlled, through contracts and negotiations, and more easily understood – reducing the complexity of the effects on overall behavior due to subtle interactions. The reduction in the amount of subtle interactions between behaviors should lead to an increase in the level of actor complexity that a designer can be reasonably expected to develop.

Both major aspects of EASE, namely the underlying computational engine and the overlying specification process, have been designed by looking at existing systems and attempting to improve *modularity*. Our previous experience suggests that modularity in agent specifications is a key to scaling up, reducing costs, improving testability and so on. The intended usage scenario for EASE makes modularity even more critical. In particular, good modularity should provide the following desirable properties: 1) rapid prototyping, 2) highly complex actors, 3) use by novices, 4) a good development process, and 5) support for development teams. Future work is intended to make the system even more usable by domain experts. EASE is currently beginning testing on site at Saab with simulation experts.

Acknowledgments

This work is supported by Saab AB, Operational Analysis division, NUTEK grants IK1P-97-09677, IK1P-98-06280, and IK1P-99-6166 and CENIIT grant 99.7.

4. REFERENCES

[1] Saab Military Aircraft. The TACSI users guide. Technical report GDIO-MI-98:356. 1995. Edition 5.2.

[2] R. Brooks. Intelligence without representation. *Artificial intelligence journal*, 47:139–159, 1991.

[3] J. Bryson. Agent architectures as object oriented design. In M. Singh, editor, *ATAL '97*. Springer, 1998.

[4] N. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous agents and MAS*, 1:275–306, 1998.

[5] H. Nwana. A perspective on software agents research. *Knowledge Engineering Review*, 1999.

[6] L. E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE RA*, 14(2), 1998.

[7] M. Wooldrige and N. Jennings. Pitfalls of agent oriented development. In *Agents '98*, 1998.

[8] J. Yen and N. Pfluger. A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation. *IEEE SMC*, 25(6), 1995.