

A User Controlled Approach to Adjustable Autonomy

N. E. Reed

Department of Information and Computer Sciences
University of Hawai'i at Manoa
Email: nreed@hawaii.edu

Abstract

This paper describes a framework for collaboration between a user and a multi-agent system to achieve adjustable autonomy. Adjustable autonomy (AA) is when the levels of autonomy of the agent system - its control over its reasoning - changes during execution due to interaction with a user or other systems. We describe a prototype agent development environment that allows users flexible on-line control over an otherwise completely autonomous agent system.

AA can improve productivity during system design, allow earlier deployment, and create a more flexible system. AA can reduce the load on instructors when used in training situations, and allow the user to aid the system when faced with unforeseen situations or problems. Examples are given of AA in simulated pilots for fighter aircraft. We found that the AA - the user's ability to modify the behavior of the agents during execution, resulted in a better, more flexible system.

1 Introduction

Virtual environments inhabited by intelligent actors are proving to be useful tools with a variety of purposes, including training [22, 9, 6], testing [7, 14] and entertainment [10]. Intelligent actors often play the roles of humans or other intelligent entities in the virtual environment. In these complex environments, it is often difficult or impossible to define an agent's behavior in advance for every situation that might occur. Allowing the user to modify the behavior of the agent during execution is one strategy that can be used to overcome this problem.

The autonomy of a system can be described as the amount of control the system has over which goals it pursues and how it pursues those goals. Autonomy can be described as a spectrum, ranging from completely autonomous at one end to completely command driven at the other [2, 3]. Adjustable autonomy (AA) means that a system's level of autonomy changes over time, i.e., the system's autonomy level increases

or decreases for some goals it is trying to achieve. The change in autonomy can be due to interactions with human operator(s), the system itself, or other systems [13, 8, 12].

AA is often a desirable capability. It may be necessary to have some way for a user to "override" the a system after it is deployed. If an unforeseen situation occurs that the system does not recognize, a disastrous failure might result from the system's action (or inaction). For example if a sensor malfunctions, the system might "blindly" trust the sensor's value and not recognize a critical situation.

We describe an agent creation environment and the user's flexible on-line control over the behavior defined by the system designer. Actors are created and defined to perform tasks within a simulation environment.

AA provides several benefits, including easier development of actors, facilitated debugging of actions, flexibility while using the system in training situations, and recovery from unforeseen problems. Adjustable autonomy may also increase the usefulness or applicability of a system.

The rest of this paper is organized as follows. The next section introduces the simulated pilot application domain. Section 3 describes the EASE agent development environment. User interaction with agents in EASE is described in Section 4, with specific examples of adjustable autonomy in Section 5. Section 6 discusses the work. The last section gives a summary and directions for future work.

2 Applications

Prototype actors have been created in two simulation environments, simulated pilots for military aircraft and soccer players. This section introduces the simulated pilot domain.

Saab Aerospace designed and produces the JAS Gripen fifth generation fighter aircraft. In addition, Saab has developed TACSI [16], a commercial high-fidelity military flight simulator. Figure 1 shows TACSI's display with three aircraft in view. Aircraft



Figure 1: The TACSI simulator display showing one agent controlled aircraft (upper) and two enemy aircraft flying over water (blue/dark) near an island.

in TACSI can be controlled by pilots or trainees sitting at cockpit consoles including a dome simulator or programmed completely in software (agents). Any combination of humans and agents can control the aircraft in each scenario. TACSI is used for pilot training, aircraft development and marketing.

Pilot actors need to appear to be intelligent and act realistically in this very complex environment [15]. If the simulated pilots are too “predictable”, pilot trainees don’t have a realistic experience in the simulator, and are thus less prepared for real situations. Using the online control enables instructors to create scenarios for several trainees, to perform the exact same maneuvers with each one, with less effort than controlling the entire plane. They may also customize scenarios and actions for each trainee, giving a much more realistic experience to each, with far less burden on the instructors.

3 EASE overview

EASE (End-user Actor Specification Environment) is an environment for creating actors in simulation environments. EASE was designed for building intelligent actors for interactive simulation environments [17, 19]. EASE is written in Java with the aim of enabling end users (domain experts) to specify the behavior of actors without the need to continuously rely on expert programmers.

EASE interfaces with a simulation environment as shown in Figure 2. EASE and the simulation environment may be on the same or multiple networked

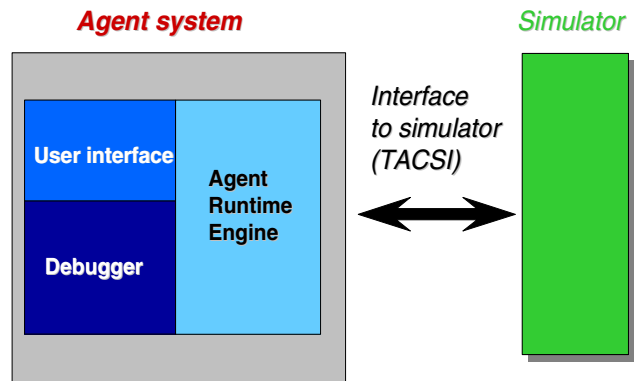


Figure 2: Interaction between EASE, the user and a simulation environment. EASE contains facilities for actor development as well as a run-time engine that interfaces with the user and the simulation environment

machines (During development and testing, both ran concurrently on the same sun workstation).

An EASE actor is composed of a multi-agent system (MAS). The system designer specifies the actor’s behavior using the tools in 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 one 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 each other to select the actual action the actor should take.

An EASE actor’s MAS is a forest of trees, as shown in Figure 3. Non-leaf agents are called *managers*, their job is to find and *contract* other agents to achieve their goals. Leaf agents are called *engineers*, as they negotiate with each other in *factories* over the actions (behavior) of the actor. The contracts correspond to goal/sub-goal relationships.

As the current goals change during a scenario, contracts are made and broken, agents are added and deleted from the trees, and therefore the structure of the forest changes. At each point in time, the trees reflect the current task breakdown of the overall actor.

The sequence of behaviors within an agent (one node in a tree) is specified with a state machines using an intuitive GUI in EASE. In each state, a manager may contract agents to accomplish its goals. During

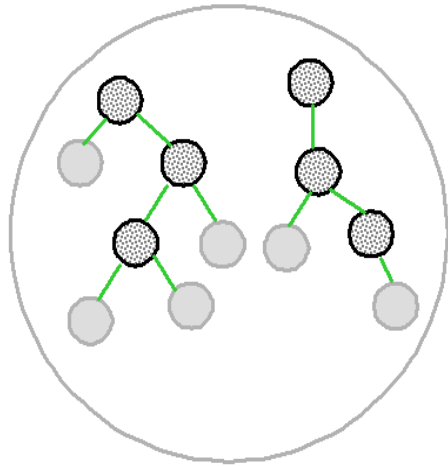


Figure 3: The structure of an EASE actor’s multi agent system. A forest of trees contains manager agents (non-leaf nodes) and engineer agents (leaf nodes). Lines between agents represent contracts.

runtime, any newly created (contracted) agents are added to the MAS in the tree below the contracting agent.

For example, we might want to specify a simple patrol agent for a pilot in TACSI. Suppose we want a plan like the one shown in Figure 4. Using EASE, this patrol mission would be created using the agent specification interface resulting in a state machine as shown in Figure 5. The “start” state of the agent is identified with a short line (upper left circle in the figure). When the agent is created, execution of the state machine starts in that state.

During execution, each agent must keep its contractor informed of its progress. The agent reports whether its goal is currently being achieved, its goal cannot be achieved or normal progress is being made towards the achievement of its goal (i.e., there is no reason to believe the goal cannot be achieved but it has not yet been achieved). The messages allow managers to make decisions based on the status of their sub-goals. The simplicity of these messages between agents makes adding to and removal of agents from the organization relatively easy, and is used for online control, as described below.

Agents can be selected for contracts specifically or based on their capabilities, using a capability matcher and descriptions of the capabilities of each type of agent. A pool of *generic agents* exists, each of which

Intended flight path of the actor

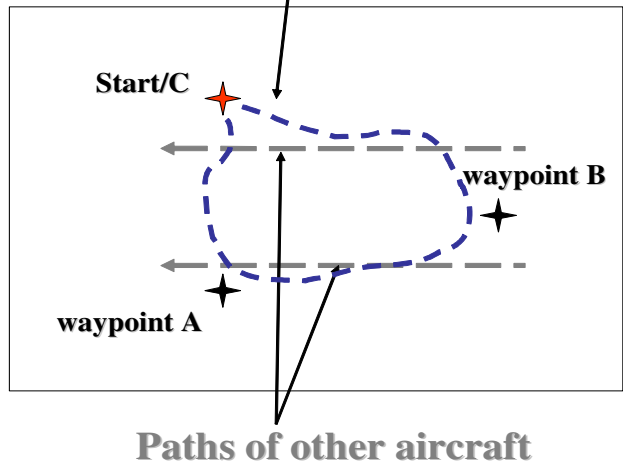


Figure 4: A simple mission flight plan for our actor showing 3 waypoints to fly between. This corresponds to the state machine shown in Figure 5. In the example to follow, 2 other (enemy) aircraft happen to appear during our patrol mission.

can be contracted to perform their specialty by an agent already part of the MAS. The generic pool effectively represents the *abilities* of the actor. If there is an agent in the generic pool capable of achieving some particular goal then the actor can achieve that goal (barring unforeseen problems in execution). If there is no agent with the capability to pursue a certain goal, then the actor cannot perform that goal.

EASE enforces a methodology for actor development that covers all stages of development, from design through to reuse[19]. The tools in EASE 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.

4 Online Control

In addition to the behavior specification tools used to design agents and their behavior, EASE includes a set of graphical tools that enable the user to monitor the state of and interact with an actor during simulations. The tools were designed to be readily understandable and usable to most users, even those

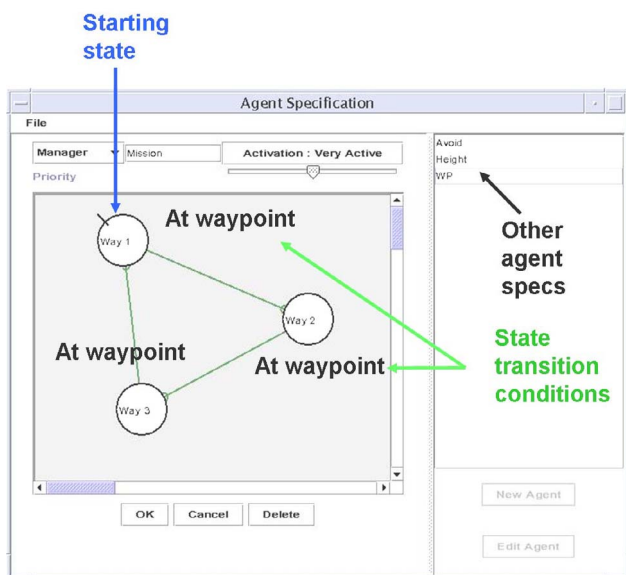


Figure 5: The state machine specification tool, showing a state machine for a simple patrol agent. The goal of the state is shown within each circle. The lines show transitions between states. This specification shows the agent repeatedly flying a triangular pattern through the 3 waypoints.

without much training in agents or programming [20]. This is typically the case for domain experts - they are expert at their specialty, not at programming.

The interfaces implemented for this project are prototypes, built to demonstrate the ideas, not for practical use. The current interfaces, albeit simple, provide the basic functionality required for the changes in autonomy desired by the user.

4.1 The Boss

The run-time support environment allows the user to monitor and *modify* agent behavior at run-time without stopping the simulation or taking over complete control of the agent. The primary interaction window in EASE is called *The Boss*.

The Boss is a tool that (among other things) shows the status of the agent hierarchy within the actor. A snapshot of *The Boss* is shown in Figure 6. The tool is nicknamed *The Boss* because it provides most of the functionality for controlling an actor. The hierarchy (or hierarchies) of agents are displayed using a collapsible tree structure. More information about each agent is shown in different colors (type of agent - manager/engineer), its name (text) and its current state

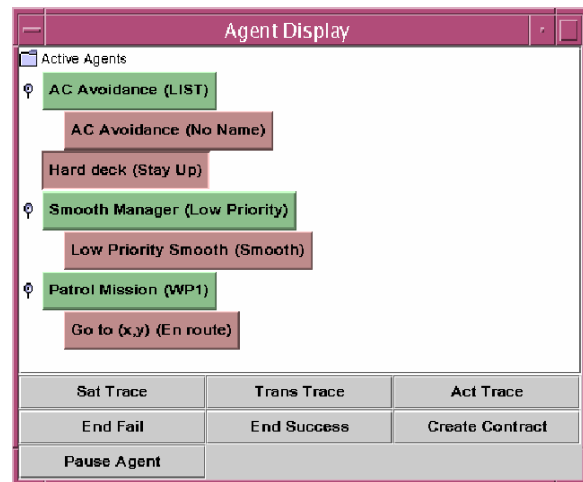


Figure 6: The *BOSS* display window showing the hierarchy of active agents (top) and control buttons (bottom). This screen shows an aircraft during a patrol mission, with agents to avoid any aircraft that are observed, avoid crashing into the ground (hard deck), fly in a smooth manner, and go to a waypoint. The 3 control buttons in the upper row activate additional windows for the user to monitor the underlying calculations producing the pilot's behavior. The lower 4 buttons are used to add, remove, and suspend agents in the hierarchy.

(text in parentheses).

The visualization gives an accurate and comprehensive picture of the agent organization. The user can extract information such as “the actor is attempting to go to waypoint A (or do X) because the mission agent (goal Y) is currently active and in a state trying to reach A (X).

The information for *The Boss* comes directly from the structure of the organization which is explicitly represented in the actor. Notice that the interface makes virtually no translation from the underlying situation to the view presented. Firstly, this means the interface was (more or less) trivial to build. Secondly, it means that under a very wide range of circumstances the interface presents an accurate picture of the underlying situation. A process that needs to do a non-trivial translation will sometimes lose or mis-interpret information. Hence, the simplicity of this visualization gives us increased confidence in its accuracy.

Actor behavior can be monitored and dynamically modified by adding, removing, or suspending agents in the multi-agent system controlling the pilot/actor as is explained in more detail below. The agent structure shown in *The Boss* (Figure 6) reflects the structure of

agents in the forest of trees described in Figure 3. Each of the agents aligned to the leftmost are the top nodes of trees. The agents contracted by each top node are shown indented underneath.

When a scenario is started, an actor is created from the currently loaded specification file. The actor executes autonomously, unless the user makes changes.

4.2 Adding agents

To add an agent to the organization the user clicks the “Create contract” button (Figure 6), then uses a dialog box to select which of the existing agent types to add. If the agent type has parameters (for example waypoint coordinates), a dialog box will appear to ask the user to enter their values, then the parameters are instantiated in the agent’s contract.

A new agent added to the MAS by the user creates a new root node (tree) in the forest. Each added agent is an added goal for the actor to pursue. The newly added goals/agents are handled in the same way as any other goals of the actor. It is possible that a whole hierarchy of agents is eventually contracted to achieve the added goal, e.g., if the user adds a manager to the organization, that manager will contract other agents in the normal manner.

In the usual case a newly added agent, or its contractees, will result in some change in actor behavior due to its involvement in negotiations. For example, if an agent for flying at some particular altitude is added to the agent organization of a simulated pilot, the aircraft might climb (or dive) to that altitude while continuing to pursue whatever other goals it has.

If the added agent’s priority is low and there are agents with higher priority and conflicting goals, the addition of the agent may have no effect on the actions of the actor because the higher priority agents “win out” in the negotiation.

If an added agent (or its contractees) has sufficiently high priority so that it has a significant say in negotiations it is possible that the achievement of other goals already in the system is delayed or never accomplished.

4.3 Removing Agents

To remove an agent from the organization, the user selects the agent in The Boss window and clicks either the “End Success” or “End Failure” button. The removal of agents from the actor has the opposite effect of adding them, as expected. When an agent is deleted, the entire subtree rooted by that agent is removed - the goal it was pursuing is removed from the set of goals the actor is pursuing.

If the removed agent is not a top-level agent (i.e. it has a manager), it is necessary for the agent to inform

its contractor that it will no longer be pursuing its assigned goal. The user decides which message is sent depending on how they want the contractor to react to the removal of the agent. In both the success and failure cases the messages that are sent are the same as would have been sent if the agent detected its own success or failure (rather than being removed by the user). The two options give the user the ability to choose the success or failure transitions of the manager (in cases where there is a difference).

The contracting agent uses the success/failure information to guide its future actions. Clearly, *success* messages indicate that the contractor should act assuming that the goal assigned to the (now defunct) contractee has been achieved. Conversely, failure messages imply that the contractor should assume that the contractee has failed and act accordingly. Precisely what the contractor will do has been specified by the designer via the use of success and failure transitions in the contractor’s state machine.

Analogous to the effect of adding of goals, the removal of goals could potentially allow previously failing goals to succeed, i.e., the removal of some engineers from a negotiation may allow other (previously lower priority) engineers to succeed where they were previously failing.

4.4 Suspending agents

Agents can also be *suspended*. The result is as if they had been removed, with the exception that the entire sub-tree can be reactivated at a later time if the user so chooses. While suspended, the agents (and all agents contracted by them) are also suspended. During the suspension, they do not participate in negotiations and thus have no effect on the behavior of the actor. Agents can be restarted, in which case they resume from the same state they were in when they were suspended. We will not discuss this facility in detail because it is functionally equivalent to removing then adding the same agent to the actor.

4.5 Changing constants

Changing constants used in calculations such as an agent’s priority, can be achieved both before and during execution (online). One limitation of this type of change is that only those constants explicitly designed in at design time can be changed at run-time. For example, if there was no *low_fuel_level* constant defined, the user cannot use that constant to change an aspect of the behavior at runtime. The constant could, however be included in a new agent specification and then would be available at runtime in the future.

The condition specification window, shown in Figure 7, allows the user to graphically specify constants

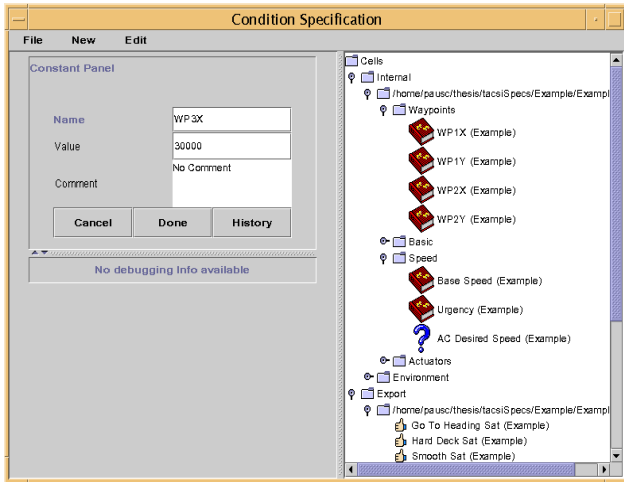


Figure 7: A snapshot of the condition specification sub-system in EASE. Functions and constants used in agent calculations can be specified and modified using this interface.

and complex functions used in the state machines to specify behavior. Additions, deletions and changes may be made online or offline and can be saved for later use.

5 Examples of User Interaction

In this section we present an extended example of an EASE actor for the simulated air-combat domain using Saab’s tactical air-combat simulator TACSI [16]. The actor’s behavior is fairly simple so we can focus on the adjustable autonomy capabilities available and the resulting changes in behavior.

First, a scenario file is loaded into TACSI and an agent specification file is loaded into EASE. Next, the simulation and EASE actor(s) are started. Figure 8 shows the central control panel in EASE with a start/stop (toggle) button. The button currently shows “Stop” since the actor has been started. The other EASE interface panels are launched by clicking the other buttons in this control panel. The name of the actor specification being used is displayed in the text field at the bottom of the window. In this case the specification is named *Example.act*.

Figure 1 shows the starting positions of the aircraft in this scenario - they are all in the air and at the same altitude. An EASE actor is controlling the aircraft labeled with a “1” in the upper left part of the screen. The other 2 (enemy) aircraft will fly from right (East) to left (West) at a fixed altitude and speed (they are



Figure 8: A snapshot of the “Start control” tool from which the actor and other tools are started.

not controlled by EASE actors in this scenario). The scenario occurs over the water near an island off the east coast of Sweden.

The actor in this example has control over the altitude, heading and speed of the aircraft. The TACSI simulator includes sub-simulators to perform the very low level details of keeping the aircraft in the air. The dynamics model built into TACSI is realistic, restricting what the aircraft can do. For example, if the actor asks for an immediate 180 degree turn, the aircraft would not be able to turn that quickly, instead it will turn as quickly as the flight dynamics and fuel use allow.

The “Watch Agents” button in the start window launches “The Boss” control interface. Figure 9 shows the initial agent organization for this scenario, consisting of three agent hierarchies (trees). At the top is the *AC Avoidance* list manager. This manager will contract agents to avoid each aircraft the actor detects.

Second from the top is an agent called *Hard deck* which is responsible for ensuring that the aircraft stays above a certain altitude. This agent has a high satisfaction value when the aircraft is above a fixed altitude. The *Hard deck* agent’s priority is high.

The final hierarchy, headed by the *Smooth Manager*, is for maintaining a “smooth” trajectory for the aircraft i.e. ensuring that turns are not too tight and the aircraft does not try to change speed or altitude too quickly. This agent results in much smoother changes than are enforced by the low-level aircraft dynamics routines in TACSI.

5.1 Adding an Agent/Goal

The agents currently active (avoid aircraft, hard deck, and smooth) do not specify a *mission* for the actor, although they provide necessary safety-related

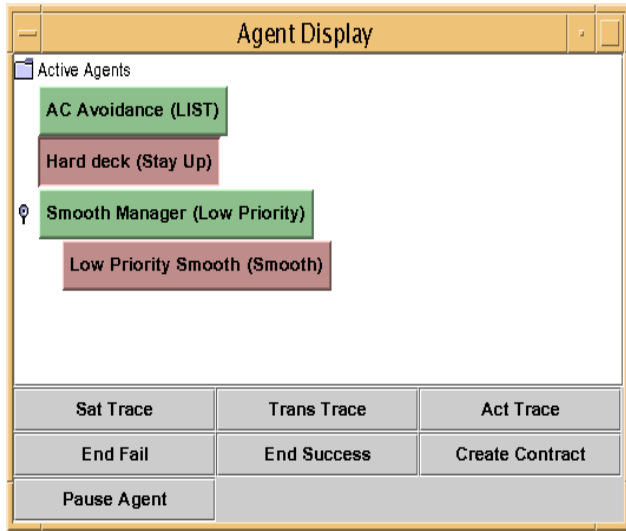


Figure 9: The Boss display showing the starting agent organization of the actor in the scenario.

goals. Figure 10 shows the user selecting a *Patrol Mission* agent after clicking on the “Create Contract” button on The Boss. The mission implemented by the *Patrol Mission* agent is to fly a triangular pattern through 3 waypoints (as shown in Figure 4). The patrol is flown indefinitely. The new *Patrol Mission* (PM) agent contracts parameterized agents to get to each of the mission’s waypoints. When a waypoint is reached, a “success” message is sent to the PM agent, which then cancels the current contract and creates a new contract with a new waypoint agent for the next waypoint.

Figure 11 shows The Boss after the *Patrol Mission* agent has been created. Now the actor has a reason to change course - to reach the first waypoint. After some negotiation, the actor turns the aircraft towards the waypoint.

At this point, an enemy aircraft is detected and agents are created automatically within EASE that avoid a collision. This does not involve user interaction, so we will skip over that part of the scenario.

5.2 Deleting an Agent/Goal

This section describes how an agent is deleted from an actor. Suppose the user now wants to take control of the mission, they can remove the existing *Patrol Mission* agent. This is achieved by selecting the agent and clicking on either “End Fail” or “End Success” buttons. In this case it does not matter which but-



Figure 10: The user has selected a *Patrol Mission* agent for addition to the agent organization.

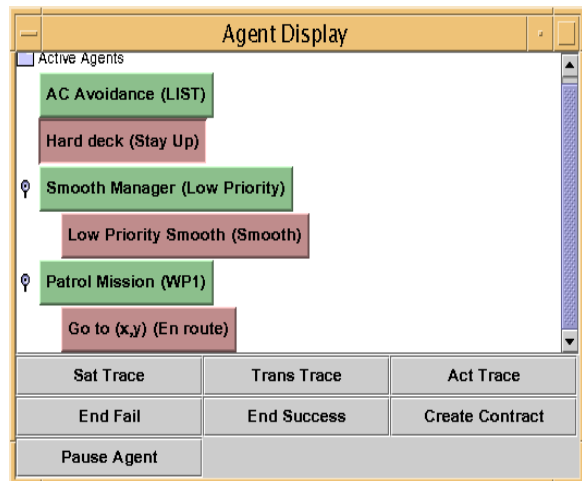


Figure 11: A snapshot of The Boss after the *Patrol Mission* agent has been contracted.

ton is used since the agent has no contractor to send a message to, anyway. Figure 12 shows a snapshot of The Boss after the *Patrol Mission* agent has been stopped.

We observe that the user did not need to plan the low level details of the aircraft’s behavior. The *AC Avoidance*, *Hard Deck* and *Smooth* agents kept performing their functionality so the user could focus only on the aspects of behavior that interested them. For example, if another aircraft had been detected it would have been avoided by the *AC Avoidance* manager without further involvement from the user.

5.3 Suspending Agents

In this section we show the effect of a particular agent on the overall actor behavior by suspending that

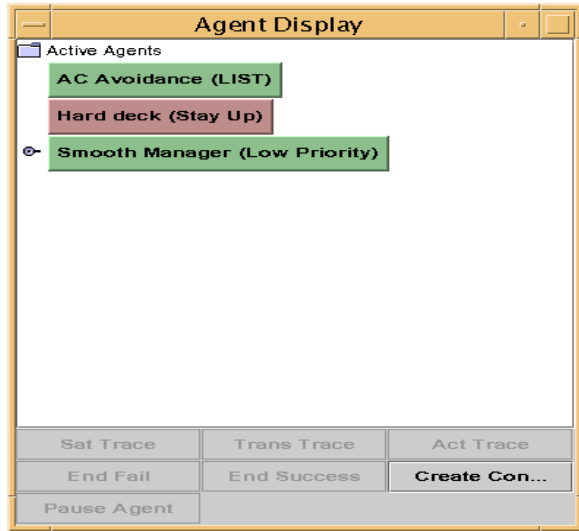


Figure 12: A snapshot of The Boss after the *Patrol Mission* agent is deleted. The Smooth Manager’s tree has been “collapsed”, thus its contractees are not divisible.

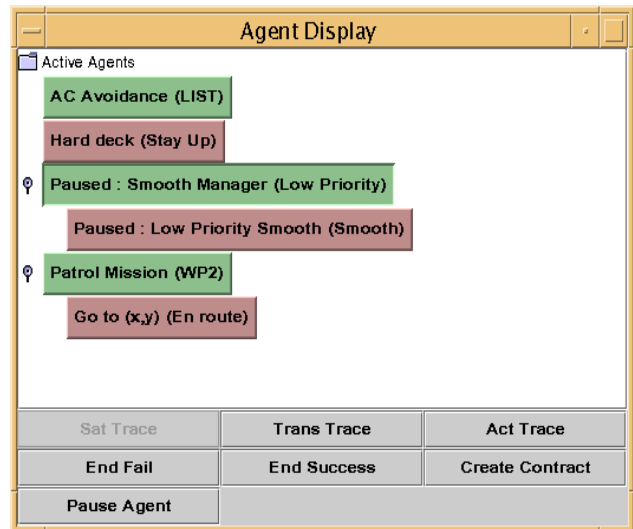


Figure 13: A snapshot of The Boss with the *Smooth Manager* agent suspended.

agent. We will suspend the *Smooth Manager*. This is accomplished by selecting the *Smooth Manager* (SM) in The Boss window and clicking “Pause Agent”. The influence of the SM agent has then been removed. Figure 13 shows that the whole hierarchy headed by the SM has been suspended. A trace of the aircraft, starting when the SM was active, then the SM is suspended and more turns are performed is shown in Figure 14. The longer, smoother turn at the bottom was made while the *SM* was active while the tighter turns at the top were made when the *SM* was suspended. In this case the effect of the *SM* is evident but not extremely strong. The designer might choose to modify the specification in the future by increasing the priority of the *SM* to make turns smoother.

The examples above demonstrate the primary ways that a user can change the actor’s behavior on-line - by adding, deleting, or suspending agents in the actor’s MAS. A user may also modify behavior by changing the constants and equations that are used in the negotiations and specifications. The interface for changing these is shown in Figure 7.

6 Discussion

EASE actors have been created and tested flying aircraft in the TACSI environment as well as playing soccer in the RoboCup simulation league [21, 18]. For

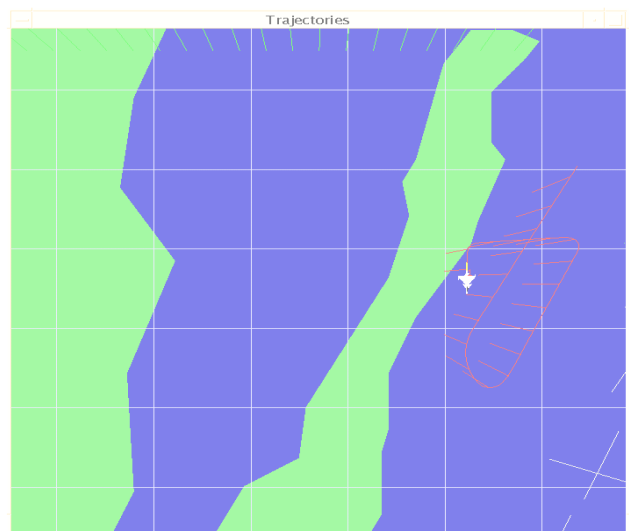


Figure 14: A trace of the aircraft’s path during the patrol. The turn at the bottom (one long turn) was made while the *Smooth Manager* was active and the tighter turn at the top was made while the *Smooth Manager* was paused.

a good overview of RoboCup research, see Burkhard, et al. [5].

During development, EASE controlled pilots were demonstrated in several scenarios to engineers and project managers at Saab. Their feedback was extremely helpful and encouraging during both development and testing. Adjustable autonomy proved useful in this domain for debugging the behavior of pilots and also for use in training scenarios.

In EASE there is no change in autonomy within the agent organization, i.e., a particular agent's autonomy relative to the other agents is constant. The autonomy of the actor relative to the user changes dynamically. Other architectures have AA between agents, e.g., [11, 1], but for EASE the AA is only between the actor and the user. Changing the priority of agents in EASE would be similar to changing the number of votes as in Barber, et al. [1].

The boss interface enables the user to modify the actor's behavior during execution, by adding, removing and suspending agents in the hierarchy. Several other information display windows are also available. It is the user's responsibility to determine what information is relevant to them at the current time. For example, the Boss display shows the entire agent hierarchy, but only some parts will be important for any particular purpose and the user needs to determine which are the important ones. Better interfaces could be designed that would call the user's attention to more important items, e.g., agents that are failing. User modeling could be used to enhance the interface and further assist the user.

The user's ability to define an agent's behavior with state machines and goal hierarchies proved to be relatively intuitive for domain experts - e.g. pilots. The ability to re-use parts of actors and modify the agent's goals during runtime proved to be useful for debugging behavior as well as creating more variety in scenarios. The online control in EASE was useful in both domains for debugging actor behavior and also for use in pilot training scenarios.

Several limitations of this approach to on-line control are evident. EASE gives the user an intuitive way to specify a broad range of behaviors, but it is not necessarily easy to specify all types of behaviors in this way. In particular, the current approach does not include planning (long) sequences of actions - it is similar to a behavior-based approach.

The user must both *understand* the actor's current behavior and be able to respond with *modifications* fast enough, for them to be of use during a simulation. Depending on the dynamic nature of the environment,

and human response times being significantly slower than software, this limits the applications.

7 Summary and Future Work

This paper describes the on-line interaction and control available to the user of a multi-agent system acting within a complex, dynamic simulation environment. The user has broad control over the autonomy of the agent at run-time. Experimental results with simulated pilots for military aircraft demonstrated the ideas.

Future work involves refining and enhancing the human-computer interaction in the system. Enhanced user interaction has the potential to enable users to more quickly understand the actor's behavior and thus be able to modify it more easily and quickly. Additional testing with other simulations, for example search and rescue would give further insight into the approach. Increasing the modes of adjustable autonomy, for example having an agent sometimes initiate a change in autonomy, would also be an interesting addition.

Acknowledgments

This work was supported by The Network for Real-Time Research and Education in Sweden (ARTES) project 0055-22, Saab Corporation's Operational Analysis Division, the Swedish National Board for Industrial and Technical Development (NUTEK) under grants 97-09677, 98-06280 and 99-6166, and the Center for Industrial Information Technology (CENIIT) under grant 99.7. Tack så mycket to the engineers and managers at Saab Aerospace in Linköping.

References

- [1] K. S. Barber, C. Martin, and R. McKay. A communication protocol supporting dynamic autonomy agreements. In *Proceedings of PRICAI 2000 Workshop on Teams with Adjustable Autonomy*, pages 1–10, Melbourne, Australia, 2000.
- [2] K. S. Barber and C. E. Martin. Agent autonomy: Specification, measurement, and dynamic adjustment. In *Autonomy Control Software Workshop, Autonomous Agents 99*, pages 8–15, 1999.
- [3] K. Suzanne Barber, Cheryl E. Martin, Nancy E. Reed, and David Kortenkamp. Dimensions of adjustable autonomy. In *Advances in Artificial Intelligence: PRICAI 2000 Workshop Reader*, volume 2112 of *Lecture Notes in Artificial Intelligence*, pages 353–361. Springer Verlag, Berlin, 2001.

- [4] Bruce Blumberg. Go with the flow: Synthetic vision for autonomous animated creatures. In *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 538–539, Marina Del Ray, CA, Feb 1997.
- [5] H.-D. Burkhard, D. Duhaut, M. Fujita, P. Lima, R. Murphy, and R. Rojas. The road to RoboCup 2050. *IEEE Robotics and Automation Magazine*, pages 31–38, June 2002.
- [6] Paul Cohen, Michael Greenberg, David Hart, and Adele Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3):32–48, 1989.
- [7] M. Craft and C. Karr. Testing future weapons systems using cgf systems. In *Proceedings of the sixth conference on computer generated forces and behavioral representation*, pages 141–150, Orlando, Florida, July 1996.
- [8] G.A. Dorais, R.P. Bonasso, D. Kortenkamp, B. Pell, and D. Schreckenghost. Adjustable autonomy for human-centered autonomous systems. In *Workshop on Adjustable Autonomy Systems at IJCAI-99*, pages 16–35, August 1999.
- [9] R. Dörner, Paul Grimm, and Christian Seiler. Agents and virtual environments for communication and decision training emergencies. In *Proceedings of the fourth international conference on Autonomous Agents, Agents 2000*, pages 50–51, 2000.
- [10] P. Doyle and B. Hayes-Roth. Agents in annotated worlds. In *Proceedings of the Second international conference on Autonomous Agents*, pages 173–180, 1998.
- [11] Christian Gerber, Jörg Siekmann, and Gero Vierke. Holonic multi-agent systems. Research Report RR-99-03, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1999.
- [12] David Kortenkamp, Robert Burridge, Peter Bonasso, Debra Schrenkenhoist, and Mary Beth Hudson. An intelligent software architecture for semi-autonomous robot control. In *Autonomy Control Software Workshop, Autonomous Agents 99*, pages 36–43, 1999.
- [13] D. Musliner and K. Krebsbach. Adjustable autonomy in procedural control for refineries. In *AAAI Spring Symposium on Agents with Adjustable Autonomy*, pages 81–87, Stanford, California, 1999.
- [14] Richard Pew and Anne Mavor, editors. *Modeling Human and Organizational Behavior*. National Academy Press, Washington, D.C., 1998. National Research Council.
- [15] Nancy E. Reed and Paul Scerri. Adjustable autonomy in simulated pilots. In *International Joint Conference on Artificial Intelligence, Adjustable Autonomy Systems Workshop*, pages 56–59, August 1999.
- [16] Saab. *TACSI - User Guide*. Gripen, Operational Analysis, Modeling and Simulation, 5.2 edition, September 1998. in Swedish.
- [17] Paul Scerri. *Designing Agents for Systems with Adjustable Autonomy*. PhD thesis, Computer and Information Sciences Department, Linköping University, December 2001.
- [18] Paul Scerri, Nancy Reed, Tobias Wiren, Kikael Lönneberg, and Pelle Nilsson. *Headless Chickens IV*, volume 2019 of *Lecture Notes in Artificial Intelligence*, pages 493–496. Springer Verlag, Berlin, 2001.
- [19] Paul Scerri and Nancy E. Reed. Creating complex actors with EASE. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Fourth International Conference on Autonomous Agents (Agents 2000)*, pages 142–143. ACM Press, June 2000.
- [20] Paul Scerri and Nancy E. Reed. Engineering characteristics of autonomous agent architectures. *Journal of Experimental and Theoretical Artificial Intelligence*, 12(2):191–212, April 2000.
- [21] Paul Scerri and Nancy E. Reed. Online control of agents using EASE: Implementing adjustable autonomy using teams. In Nancy E. Reed, editor, *Workshop on Teams with Adjustable Autonomy, Sixth Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000)*, pages 25–34, August 2000.
- [22] Milind Tambe, W. Lewis Johnson, Randolph Jones, Frank Koss, John Laird, Paul Rosenbloom, and Karl Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1):15–39, Spring 1995.