

Making Adjustable Autonomy Easier with Teamwork

Paul Scerri and Nancy E. Reed

Department of Computer and Information Science
Linköping University, SE-581 83 Linköping, Sweden
pausc@ida.liu.se | nanre@ida.liu.se

Abstract. Adjustable Autonomy (AA) is the name given to a variety of approaches to the task of giving outside entities the ability to change the level of autonomy of agents in an autonomous system. The idea presented in this paper is to leverage the flexibility and robustness of agent teams to develop flexible and safe AA mechanisms. We argue that the fundamental properties of teamwork are very useful for providing the robustness to change and flexibility that an implementation of AA requires. We present the EASE system which leverages teamwork as the basis for a powerful and robust AA system for intelligent actors in interactive simulation environments.

1 Introduction

Despite our best efforts, so called “intelligent” actors sometimes don’t do precisely what we expect or intend them to do. Coffee lounges are full of anecdotes of actor designers watching in horror as at a critical moment in an important demonstration an “intelligent” actor does something extremely stupid. Giving designers some control over actors at runtime will increase their utility - as well as reducing stress [Blumberg and Galyean, 1995]. In actors deployed over long periods of time, with safety critical missions, perhaps on spacecraft, it may be essential that the actor’s behavior can be modified without restarting it [Dorais, 1998, Schooley *et al.*, 1993, Kortenkamp *et al.*, 2000].

Adjustable Autonomy (AA) is the name given to a variety of approaches to achieving the goal of allowing the parts of a system to change levels of autonomy [Kortenkamp *et al.*, 1999b, Musliner and Krebsbach, 1999, Reed, 2000, Hexmoor, 1999]. In this paper we are primarily concerned with mechanisms giving humans some control over autonomous actors in simulation environments. This means giving the user the ability to change aspects of the agent’s behavior at runtime. The behavior should be able to be changed at all levels of abstraction and at any time during the simulation. The more flexibility the AA gives to a user the better the AA serves its purpose.

However, AA is far from trivial to build into an actor. Two general approaches are immediately obvious. Firstly, at design time the designers could work out the different controls the user will need at runtime and build them into the actor.

This approach is taken in systems like HCSM [Cremer *et al.*, 1995] and Improv [Perlin and Goldberg, 1996]. An advantage of this approach is that the designers can ensure a priori that the actions the user takes at runtime will be safe and sensible. A limitation of the approach is that the designer needs to correctly identify all the actions the user might want to take, a priori. This is problematic as it seems that one of the important uses of AA is for graceful handling of situations that were *unexpected* at design time.

A second general approach to AA is to augment the agents with mechanisms that allow users to flexibly manipulate the internals of the actor. This is the approach taken by [Kortenkamp *et al.*, 1999b] and [Blumberg and Galyean, 1995] among others. Such mechanisms, if done well, can provide the user with an almost unlimited range of options, allowing them to make whatever behavior changes are required at runtime. One downside of such approaches is that, because the designer gives up some control over the internal running of the actor, it becomes difficult to guarantee that the actor's behavior will be reasonable, safe or graceful. Even carefully designed specifications can easily fall apart when a user starts making ad hoc changes.

The general approach of providing mechanisms to allow a user to manipulate the internal structure of an actor is appealing because it allows great flexibility. However, the promise can only be realized if the actor can handle fairly ad hoc changes to its structure robustly.

Most agent systems are not designed to allow ad hoc changes to their composition. However, there is one type of system that *is* specifically designed to be flexible and robust to changes to its components – multi-agent systems (MASs) [Tambe, 1996]. When the agents work together as a *team* towards common goals they can be especially robust and flexible. Part of the reason for the flexibility is that interactions between the components (i.e., agents) of the system are at an abstract level. The relationships between agents are explicitly represented and understood so when one part changes (or breaks) the rest of the team can reconfigure smoothly around the changed or broken agent.

We leverage the flexibility and robustness of agent teams to develop flexible and safe AA mechanisms. The agent teamwork implements the same type of guarantees, e.g., safe and reasonable, that a designer might implement if developing hard-coded AA controls - but in a way that is flexible at runtime. In particular a single agent is composed of a complete MAS. (For clarity we refer to the overall agent (composed of a MAS) as an actor, and the components of the actor (team members in the MAS) as agents.) The AA mechanism gives the user ad hoc control over the MAS. The teamwork between the agents in the MAS helps make the overall behavior “graceful” and safe.

A prototype of this idea has been implemented. The real-time control prototype, nicknamed “The Boss” is a sub-system of an actor specification system called EASE (End-user Actor Specification Environment) [Scerri and Reed, 2000b]. EASE is used to develop actors which are internally controlled by a multi-agent system (MAS). The Boss works by adding agents to, removing agents from or suspending agents in the multi-agent system which

in turn dictates the behavior of the actor. These facilities give the user a large amount of control over the actor at runtime. Importantly the user is not required to take back complete control of the actor, only manipulate those aspects of the behavior that they want or need to change. Because the addition or removal of agents is done in accordance with the team conventions and within the team model, the overall behavior of the actor usually continues to be reasonable while the user makes their desired changes. That is, the normal teamwork mechanisms serve to provide the robustness required in the face of these ad hoc changes.

2 EASE

EASE is an integrated graphical development system for specifying and running intelligent actors for simulation environments. Specific design tools support all aspects of development from information processing specification through to mission specification. For more details on EASE see [Scerri and Reed, 2000b], in this paper we describe only enough of EASE to support the AA.

An actor created with EASE has a complete multi-agent system that performs the task of action selection. The basic idea is conceptually similar to the “Society of Mind” ideas of Minsky [Minsky, 1988] and that of *behavior-based architectures* [Mataric, 1994]. Each agent in the system is responsible for a specific aspect of the actor’s overall behavior. For example, in a simulated pilot actor there might be one agent for avoiding the ground and another for heading towards some way point. The teamwork is fairly simple – the agents form *contracts* with each other in a hierarchical structure. Contracts correspond closely to the idea of delegation [Falcone and Castelfranchi, 1999]. The contracted agent is assigned a specific aspect of the contractor’s task. For example, a contractee of a *Safety* contractor might be responsible for avoiding a particular aircraft. The contracts, including the specific agent to be contracted, are determined at design time by the designer. Coupled with the notification of success or failure and negotiation amongst agents, the contracts form the core of the teamwork mechanisms.

Agents at the bottom of hierarchies in the MAS are directly involved in negotiations about the actor’s actions. Each agent involved in the negotiation has a function which it uses to decide its *satisfaction* with different proposed actions for the actor. A *factory* administers the negotiation. A factory continuously selects values from the domain of possible actions and suggests the values to the agents in the negotiation. The agents calculate their satisfaction value for the suggested action and return it to the factory. The factory compares the responses of the agents to the new suggestion and their responses to the previous most favored suggestion. The factory keeps a record of the negotiation and periodically executes the favored action. The factory’s algorithm for choosing between suggestions is an anytime version of a fuzzy behavior fusion algorithm [Pirjanian and Mataric, 1999]. A snapshot of the Negotiation Viewer tool showing the progress of a negotiation is shown in Figure 1.

The basic idea is that each engineer argues selfishly (i.e., it argues for actions that will lead to its goal being achieved) and the negotiation protocol tries to ensure that a fair (i.e., taking into account all the agents wishes) outcome is reached. The *priority* level of an agent affects how much weight the agent has in the negotiation. The way such a process leads to good overall decisions is described mathematically in Ossowski [Ossowski and García-Serrano, 1999]. The agent's *satisfaction* with an action is proportional to the relative reduction in the distance between the current state and its ideal state(s), that can be expected from that action as compared to other possible actions. So actions that are expected to lead to the biggest reduction in distance to the ideal states will *completely satisfy* the engineer. On the other hand, any actions expected to increase the distance to ideal states will *completely dis-satisfy* the engineer. All other actions, i.e., those that are expected to reduce the distance to the ideal state(s) but not so much as some other actions will satisfy the engineer to a proportional level.

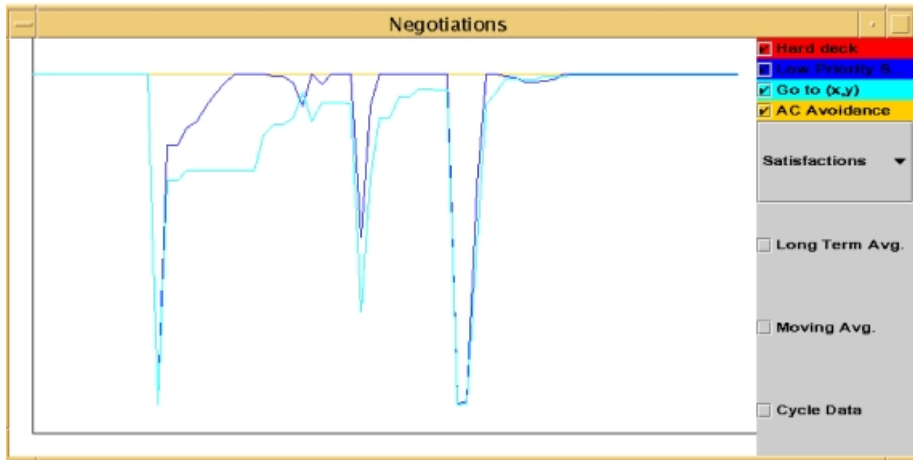


Fig. 1. A snapshot of the Negotiation Viewer showing each agent's satisfaction levels during the successful avoidance of an aircraft. Four agents are involved in the negotiation. Two of the agents, "Hard deck" (red) and "AC Avoidance" (yellow) are always completely satisfied (100% – top line). The "Go to waypoint" agent (light blue) is unsatisfied more than the "Smooth" agent (dark blue) (both lines go near the bottom of the window).

In the current EASE prototype, agents are internally controlled by single-layered state machines. Each state in the state machine may be a *success* state, a *failure* state or a regular (intermediate) state. A designer should design the agent so that when it is in a success state, it is achieving (or has achieved) the task assigned to it and when it is in a failure state, it is unable to achieve (temporarily or permanently) its assigned task. The success or failure information is

communicated to the contractor agent which can use the status information in its decision making. Figure 2 shows a snapshot of the end-user tool for creating state machines.

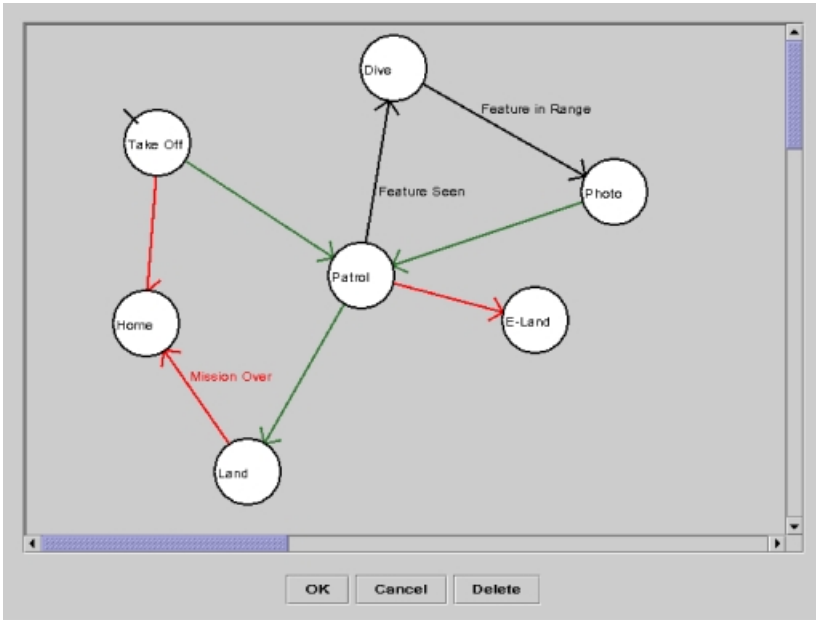


Fig. 2. The end-user state machine specification tool, showing a specification of a state machine for a simple patrol agent. Circles represent states, while arrows represent transitions. The state in the top left corner, i.e., take-off, with an extra line is the start state. Success transitions are lightly colored while normal transitions are annotated with the condition for their traversal.

Two key aspects of the EASE agents’ team behavior facilitate effective AA. Firstly, an agent informs its contractor of the success or failure of its task. This allows contractors to react intelligently to the success or failure of sub-goals and take appropriate subsequent actions. The contractee informs its contractor whether its goal is currently being achieved (or maintained), the goal cannot be achieved, or normal progress is being made towards the achievement of the goal (i.e., there is no reason to believe the goal cannot be achieved but it is not yet achieved). These fairly abstract messages are the only communication that occurs between agents once a contract is made. The messages allow contractors to make decisions based on the status of sub-goals. Although some aspects of the organization’s functioning might be improved if more status information were shared, the simplicity of the interconnections between agents makes the addition and removal of agents to and from the organization easier. Being able to add and

remove agents easily is critical when the *user* wants to add and remove agents, i.e., add or remove goals, at *runtime*.

At runtime the AA utilizes the success and failure messages to invoke the normal team mechanisms of the actor to smoothly integrate the changes specified by the user. In particular, when the user breaks a contract, i.e., removes an agent, the same messages are passed to the agent's contractor as if the agent "normally" failed or succeed. Hence, when the user takes an action the contractor agent has the required mechanisms to handle the situation and move on appropriately. This means that although the user is making "ad hoc" actions, the designer can, a priori, design the specification in such a way that the overall behavior of the actor should remain within reasonable bounds.

The second key mechanism of the team framework that facilitates effective AA is the low level negotiation mechanism. Behavior based architectures, like EASE, are intrinsically robust to failure [Goldberg and Mataric, 2000, Parker, 1998], due primarily to the flexibility of their behavior arbitration mechanisms. All "horizontal" interaction between agents, i.e., goal conflict resolution, takes place within the negotiation framework. The negotiation algorithm, being an adaptation of a behavior fusion algorithm, is well suited to having agents added and removed at unpredictable times. This allows agents to be removed from the MAS without serious disruption to other parts of the behavior.

2.1 Example

In this section a detailed example is given of the specification and running of a simple EASE actor. This example was demonstrated at Agents 2000 [Scerri and Reed, 2000a]. In the next section, this example is used to show how a user might intervene and take control, modifying the originally specified behavior of the actor online.

The example actor we describe is a pilot in an air-combat simulation. When the simulation begins, the aircraft is in the air. The pilot should fly the aircraft through three way points and then to a home position. Any other aircraft in the air should be avoided. For simplicity and clarity we consider the case of an aircraft with only two degrees of freedom: heading and altitude.

A design-time view of the agent hierarchy is shown in Figure 3. The *mission* agent has four states. In each of its states, the mission agent contracts a single *way point* agent. State transitions between the states of the *mission* agent occur when the *way point* agent it contracts reaches its way point (the *way point* agent will go into a success state and inform the mission agent of its success). There are three instances of a *way point* agent. Each of them has two states, a start state (fly) and a success state (done). In the start state the agent will negotiate using a satisfaction function that gives high satisfaction for heading suggestions toward the way point and low satisfaction for heading suggestions that are away from the way point. The transition between the states will occur when the agent reaches the way point.

The *smooth* agent, responsible for keeping the aircraft on a relatively straight course, has only a single state. In that state it has a satisfaction function that

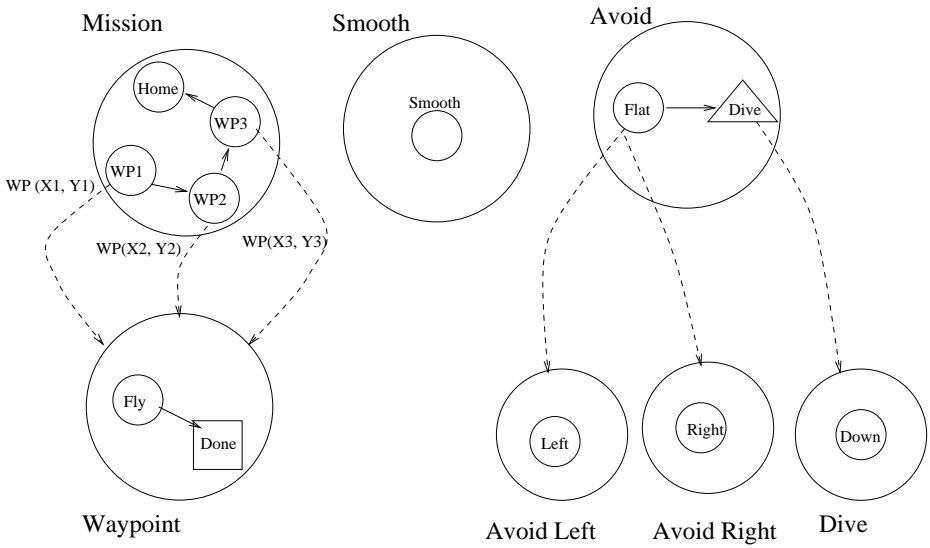


Fig. 3. The mission design specification for the example. There are three agents when the actor starts: “Mission”, “Smooth”, and “Avoid” (shown at the top of the figure). Contracts between agents are shown as dashed lines and the state machines are shown inside the agents.

prefers headings that are close to the current heading, i.e., it prefers slow turns, indirectly keeping the line of the aircraft smooth. This agent has a lower priority so that if a safety critical action cannot be done simultaneously with keeping the heading smooth the smooth agent will be overridden.

An *avoid* agent is responsible for avoiding a single aircraft (a special agent called a *list manager agent* is responsible for contracting one avoid agent for each detected aircraft). The satisfaction function in the starting state of an avoid agent is very simple. It returns high values when suggestions are made that lead to a relative heading of greater than 30 degrees from the obstacle aircraft and satisfaction values of zero otherwise. If the aircraft gets within a set distance of the obstacle aircraft a transition is made into the second state. The second state is a failure state. In this state the satisfaction function will argue for altitude values that will lead the aircraft to dive. In this state the agent’s priority is high, giving it the most weight in the negotiations.

When the actor starts there are three agents active, the *mission* agent, *smooth* agent, and *avoid* agent (see Figure 3). As soon as the actor starts, the *mission* agent will contract a *way point* agent to go toward the first way point. When the aircraft reaches the first way point, that *way point* agent will transition to its success state. Because the *mission* agent has a transition dependent on contractee success it will transition to its next state. In the next state, it contracts another *way point* agent to get it to the next way point. As soon as another

aircraft is detected, an *avoidance* agent would be contracted specifically to avoid the detected aircraft. One such agent will be contracted for each aircraft detected.

3 Online Control of Agents

In the previous sections, the multi-agent system (MAS) that provides the action selection mechanism for an actor was described. In this section, the tools that allow runtime manipulation of the MAS, and hence, runtime control of the actor, are described. The tools can be separated into two categories, tools for observing the behavior of the MAS and tools for manipulating the MAS (to change the actor's behavior).

3.1 Observing the Actor's Multi-agent System

The main tool for observing the MAS is "The Boss" shown in Figure 4. A familiar tree layout shows all the currently active agents. Each node shows the name and current state of one of the agents. The layout of the tree reflects the hierarchical arrangement of contractor and contracted agents in the MAS. It is easy to see from the tree which agent is contracted by which other agent and, more generally, how a particular abstract task is currently broken down into parts and the overall state of the actor. Different types of agents are shown in different colors. Agents negotiating directly with a factory (at leaves of the hierarchy) are shown in red and agents contracting other agents (non-leaves in the hierarchy) are shown in green.

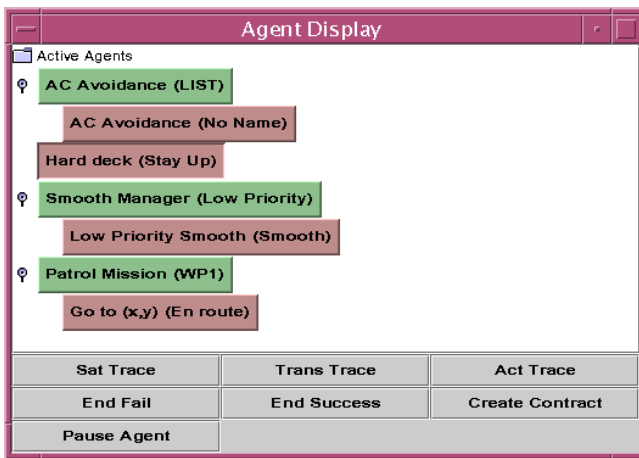


Fig. 4. The Boss display lists all active agents in an actor (top) and allows tracing, termination and suspension of a selected agent, and the creation of new agents (bottom buttons).

The top three buttons in the bottom of The Boss window (Figure 4) allow the user to view different aspects of the system. For agents involved in negotiations, a snapshot of the satisfaction function can be shown (see Figure 5). The snapshot shows, in great detail, the calculation the agent has performed in order to produce its satisfaction value for the current factory suggestion. For all the agents, traces of the calculations of their priority and state transition conditions can also be shown. This mechanism was designed primarily for debugging purposes, i.e., the designer can check the calculations at runtime to determine why agents are performing each action, but this information also aids the AA. For example, the calculation traces may allow a user to detect early on that the agent is about to do something undesirable or understand why it is doing something incorrectly.

```

Satisfaction Trace : Goal Kicker
Update
Evaluating satisfaction /home/pausc/thesis/robocupSpecs/EASESpecs/Test/Kicking.cndKick Goal S
using factory Action
Calculation named Kick Goal S
Getting sensor function : Kick to Goal S
Factory suggestion SUGGESTIONAction#Kick#Power evaluates to : 0.0
Constant Zero evaluated : 0.0
Executing Plus with 0.0 and 0.0 result is 0.0
Factory suggestion SUGGESTIONAction#Kick#Direction evaluates to : 0.0
Constant Zero evaluated : 0.0
Executing Plus with 0.0 and 0.0 result is 0.0
Sensor Kick to Goal S returned : 3.0
Calculation /home/pausc/thesis/robocupSpecs/EASESpecs/Test/Kicking.cndKick Goal S returns 3.0
Satisfaction value is : 3.0

```

Fig. 5. The Calculation Trace window shows the details of the calculations one of the agents is performing. Each line shows the result of computing the value of one cell. The labels of the cells are printed with the corresponding intermediate results. The bottom line in the window shows that the final satisfaction value calculated is 3.0.

3.2 Manipulating the Actor's Multi-agent System

There are three ways that the user can manipulate the MAS: by breaking existing contracts, by creating new contracts and by suspending active agents. Each of the mechanisms is accessed through the same interface shown in Figure 4 (the same interface used to observe the agents as described in the previous section).

To break a contract between contractor and contractee, the user selects the agent and then selects either the “end fail” button or the “end success” button at the bottom of the window. The agent whose contract is broken ends the contracts of all agents it contracted. That is, breaking a contract with an agent that has contractees results in the whole hierarchy of agents headed by that agent being terminated.

If the “end failure” button was selected the selected agent sends a message to the contractor agent informing the contractor that its contractee has failed. Conversely, if the “end success” button is used a message indicating success is sent from the agent being terminated to its contractor. The contractor handles the termination of the contract as it would handle any success or failure of a contractee, i.e., by taking the same actions that it would have taken if the contractee had really succeeded or failed.

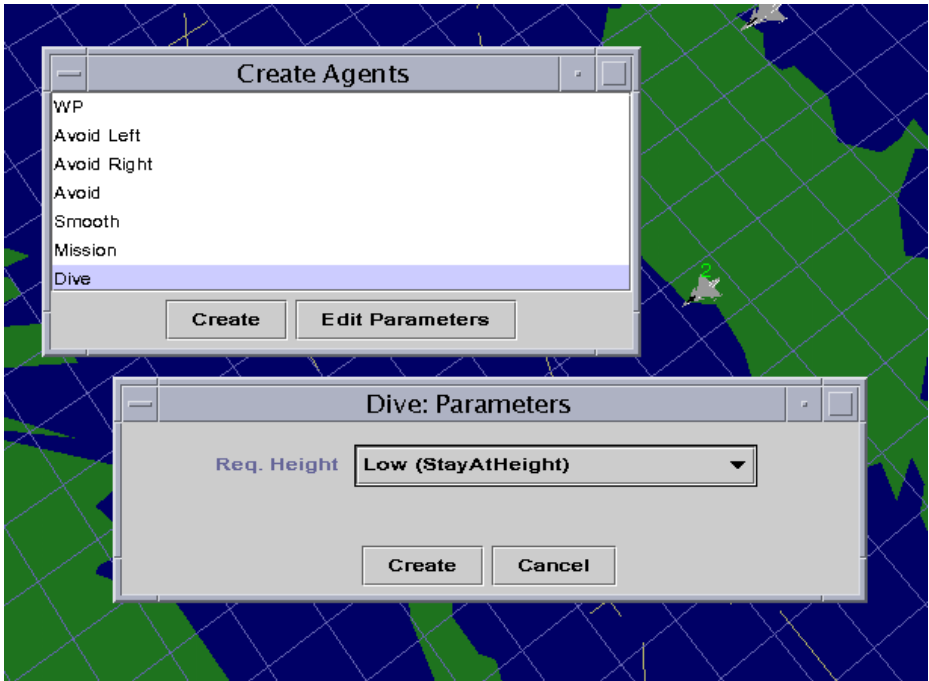


Fig. 6. Creation of new agents. In the example, the user wants to make the aircraft dive to a specified height. The user selects “Dive” for a dive agent in the upper window and then selects the “create” button. This leads to the lower window opening for the user to enter the dive altitude, in this case the user enters the parameter *low* (from the file “StayAtHeight”).

For example, consider the actor specification given in the previous section. If a user decided that the aircraft should skip the first way point and go directly to the second, this could be achieved by selecting the *way point* agent (contracted by the *mission* agent) then selecting the “end success” button. The *way point* agent will leave the negotiations and inform the contractor (the *mission* agent) of the successful completion of its task. When the *mission* agent receives the

message it will make a success state transition and contract a new *way point* agent for going to the second way point.

The second way a user can manipulate the MAS is by contracting agents themselves, i.e., by adding new agents to the MAS. Figure 6 shows another EASE tool, this tool allows agents to be created and contracted by the user. The “Create Agents” window comes up when the “create contract” button in the “Boss” window is selected. To create an agent, the user selects the agent’s name and clicks on the “create agent” button. The agent will pursue a particular goal in the actor. If the agent contract has uninstantiated parameters (for example the location of a way point) a dialog box pops up allowing the user to instantiate appropriate values. In the example shown in the figure a *dive* agent has been created and the user is instantiating the parameter “Req. Height” (indicating the level to dive to) with the value “Low” (which has been defined as some value in the file “StayAtHeight”). The created agent becomes a part of the MAS and, just like other agents, contracts other agents or enters into negotiations over actor actions. Newly created agents are always top-level agents.

The third type of control the user has is the ability to suspend (pause) an agent within the actor. This is done by selecting the agent and then selecting the “pause agent” button shown at the bottom of The Boss tool (see Figure 4). All agents contracted, directly or indirectly (i.e., contracted by contractees), by the suspended agent are also suspended. No status messages are sent to the contracting agent of the suspended agent, who assumes that the execution of the task is ongoing. The “pause” button works as a toggle, i.e. a suspended agent can be restarted by selecting it again and clicking “pause agent”.

4 Evaluation

AA is difficult to usefully test in a structured way because AA is designed to handle situations unanticipated by the designer – experiments created by a designer cannot test things they did not anticipate. Any structured experiments will likely be biased towards the features developers have included in the system. Hence, the best testing of AA is use in real world projects by real users. This is one approach being taken for the evaluation of EASE.

The original domain that EASE was used in was air-combat simulation with Saab’s TACSI system [Saab, 1998]. During one early demonstration of EASE the potential for genuinely useful AA was unexpectedly tested. The author was demonstrating a simple air-combat mission to engineers from Saab Aerospace. Mid-way through the demonstration it became clear that there was an error in the aircraft avoidance specification created for that demonstration and that a mid-air collision was likely. Realizing this, a “height” agent was temporarily contracted which caused the EASE controlled aircraft to change altitude, avoiding the imminent collision. The AA turned a potentially disastrous demonstration into a success. The most pleasing part of the incident was that the AA system was able to handle a clearly unexpected situation involving a newly introduced bug in another part of the system.

EASE was also used in the development of a RoboCup simulation team [Noda, 1995]. During the development process for the Headless Chickens IV RoboCup football team [Scerri *et al.*, 2001], AA was extensively used. The information required for doing AA reasoning is similar to that required for debugging, hence the same interfaces used to present information for AA also provided useful information for debugging. However, the ability to see inside the agent for debugging was a small bonus compared to the ability to change the players' behavior while they ran. The functionality provided by the AA was very useful for experimenting with the behavior of the agent. The task of setting up specific test scenarios was made significantly easier because the agent could be manipulated at runtime, instead of the more standard process of stopping the game and changing the player specifications. Furthermore, players with considerable gaps in their functionality, e.g., not stopping when a goal was scored or not listening to referee calls, could still be effectively tested because the developer could come in and "help out" via AA, when the player encountered a situation it did not yet have the functionality to handle.

Other application areas, including disaster management simulations, command and control training and network management simulation are currently being investigated using EASE.

5 Conclusions

We used a team infrastructure to implement a flexible, robust AA mechanism for controlling actors in simulation environments. The user manipulates a multi-agent system controlling the actor, thereby changing the behavior of the actor.

The team infrastructure brings flexibility and adaptability to the architecture, allowing it to quickly adapt to changes made by the user. When the user changes one part of the multi-agent system, the team mechanisms coordinate to produce the new behavior. This means that the user needs to pay less attention to the details of the changes they intend to make, thus making their task easier. Furthermore, the team style of partitioning the functionality of the system often makes it easier for the user to isolate the aspects of the actor's behavior that they want to observe and possibly change. Utilizing a flexible team infrastructure as the basis for AA seems to offer significant promise as a mechanism for making the AA more robust and easier to build.

Acknowledgments. This work is supported by The Network for Real-Time Research and Education in Sweden (ARTES), Project no. 0055-22, Saab Corporation, Operational Analysis Division, the Swedish National Board for Industrial and Technical Development (NUTEK) under grants IK1P-97-09677, IK1P-98-06280 and IK1P-99-6166, and the Center for Industrial Information Technology (CENIIT) under grant 99.7.

References

- [Blumberg and Galyean, 1995] Bruce Blumberg and Tinsley Galyean. Multi-level control of autonomous animated creatures for real-time virtual environments. In *Siggraph '95 Proceedings*, pages 295–304, New York, 1995. ACM Press.
- [Cremer *et al.*, 1995] James Cremer, Joseph Kearney, and Yiannis Pangelis. HCSM: A framework for behavior and scenario control in virtual environments. *ACM Transactions on Modeling and Computer Simulation*, pages 242–267, 1995.
- [Dorais *et al.*, 1998] G. Dorais, R. Bonasso, D. Kortenkamp, B. Pell, and D. Schreckenghost. Adjustable autonomy for human-centered autonomous systems on mars. In *Proceedings of the first international conference of the Mars society*, pages 397–420, August 1998.
- [Falcone and Castelfranchi, 1999] R. Falcone and C. Castelfranchi. Levels of delegation and levels of help for agents with adjustable autonomy. In *Proceedings of AAAI Spring symposium on agents with adjustable autonomy*, pages 25–32, 1999.
- [Goldberg and Mataric, 2000] D. Goldberg and M. Mataric. Robust behavior-based control for distributed multi-robot collection tasks. Technical Report IRIS-00-387, Institute for Robotics and Intelligent Systems, University of Southern California, 2000.
- [Hexmoor, 1999] Henry Hexmoor, editor. *Workshop on Autonomy Control Software. Autonomous Agents 1999*, May 1999.
- [Kortenkamp *et al.* 1999a] D. Kortenkamp, G. Dorais, and K. Myers, editors. *Proceedings of IJCAI99 Workshop on Adjustable Autonomy Systems*, August 1999.
- [Kortenkamp *et al.*, 1999b] David Kortenkamp, Robert Burridge, Peter Bonasso, Debra Schrenkenghoist, 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.
- [Kortenkamp *et al.*, 2000] D. Kortenkamp, D. Keirn-Schreckenghost, and R. P. Bonasso. Adjustable control autonomy for manned space flight. In *IEEE Aerospace Conference*, 2000.
- [Mataric, 1994] Maja Mataric. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, 1994.
- [Minsky, 1988] Marvin Minsky. *The Society of Mind*. Simon and Schuster, 1988.
- [Musliner and Krebsbach, 1999] 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.
- [Noda, 1995] Itsuki Noda. Soccer server: A simulator of RoboCup. In *Proceedings of AI Symposium '95*, Japanese Society for Artificial Intelligence, December 1995.
- [Ossowski and García-Serrano, 1999] S. Ossowski and A. García-Serrano. *Intelligent Agents V : Agent Theories Architectures and Languages*, chapter Social Structure in Artificial Agent Societies: Implications for Autonomous Problem Solving Agents, pages 133–148. Springer, 1999.
- [Parker, 1998] Lynne E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.
- [Perlin and Goldberg, 1996] Ken Perlin and Athomas Goldberg. Improv: A system for scripting interactive actors in virtual worlds. *Computer Graphics*, 29(3), 1996.
- [Pirjanian and Mataric, 1999] Paolo Pirjanian and Maja Mataric. A decision theoretic approach to fuzzy behavior coordination. In *Proceedings, IEEE International Symposium on Computational Intelligence in Robotics & Automation (CIRA-99)*, Monterey, CA, Nov. 1999.

- [Reed, 2000] N. Reed, editor. *Proceedings of PRICAI Workshop on Teams with Adjustable Autonomy*, Melbourne, Australia, 2000.
- [Saab, 1998] Saab. *TACSI - User Guide*. Gripen, Operational Analysis, Modeling and Simulation, 5.2 edition, September 1998. in Swedish.
- [Scerri and Reed, 2000a] P. Scerri and N. Reed. Real-time control of intelligent agents. In *Technical Abstracts of Technical Demonstrations at Agents 2000*, 2000.
- [Scerri and Reed, 2000b] Paul Scerri and Nancy Reed. Creating complex actors with EASE. In *Proceedings of Autonomous Agents 2000*, pages 142–143, 2000.
- [Scerri *et al.*, 2001] Paul Scerri, Nancy Reed, Tobias Wiren, Kikael Lönneberg, and Pelle Nilsson. Headless Chickens IV. In Peter Stone, Tucker Balch, and Gerhard Kraetschmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 493–496. Springer Verlag, Berlin, 2001.
- [Schooley *et al.*, 1993] L. Schooley, B. Zeigler, F. Cellier, and F. Wang. High-autonomy control of space resource processing plants. *IEEE Control Systems Magazine*, 13(3):29–39, 1993.
- [Tambe, 1996] Miland Tambe. Teamwork in real-world, dynamic environments. In *Proceedings of the Second International Conference on Multi-agent Systems*, Kyoto, Japan, 1996.