Automated Intelligent Pilots for Combat Flight Simulation

Randolph M. Jones, John E. Laird, and Paul E. Nielsen
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109-2110
{rjones, laird, nielsen}@umich.edu

Abstract

TacAir-Soar is an intelligent, rule-based system that generates believable "human-like" behavior for military simulations. The innovation of the application is primarily a matter of scale and integration. The system is capable of executing most of the airborne missions that the United States military flies in fixed-wing aircraft. It accomplishes this by integrating a wide variety of intelligent capabilities, including reasoning about interacting goals, reacting to rapid changes in real time (or faster), communicating and coordinating with other agents (including humans), maintaining situational awareness, and accepting new orders while in flight. The system is currently deployed at the Oceana Naval Air Station WISSARD, and its most dramatic use to date was in the Synthetic Theater Of War 1997, an operational training exercise consisting of 48 straight hours and approximately 700 fixed-wing aircraft flights, all flown by instances of the TacAir-Soar system.

In 1992 we began development of a software system that emulates the behavior of military personnel performing missions in fixed-wing aircraft (Tambe et al, 1995). The general goal is to generate behavior that "looks human", when viewed by a training audience participating in operational military exercises. The resulting rule-based system, called TacAir-Soar, is currently deployed at the WISSARD training facility in the Oceana Naval Air Station, which is administered by BMH Associates, a collection of active and retired military aviators who also served as our primary sources of subject-matter expertise. As the system has developed, it has taken part in a number of tests, technology demonstrations, and operational training exercises. Its most dramatic use to date was in the Synthetic Theater Of War 1997 (STOW '97)/United Endeavor Advanced Concept Technology Demonstration, held at the end of October, 1997. STOW '97 was an operational training exercise consisting of 48 straight hours and approximately 700 fixed-wing aircraft flights, all flown by instances of the TacAir-Soar system.

TacAir-Soar relies on mature intelligent systems technology, including a rule-based, hierarchical representation of goals and situation descriptions. The innovation of the system lies in the large-scale integration of a number of intelligent capabilities in a complex domain. The system does not just model a small set of tasks pertinent to military fixed-wing missions; it generates appropriate behavior for *every* such mission routinely used by the the US Navy, Air Force, and Marines; the UK Royal Air Force; and "opponent forces" in full-scale exercises. In addition to reasoning about complex sets of goals, the system coördinates and communicates with humans and other automated entities. The system must generate its behavior in real time (and sometimes faster). It must also integrate seamlessly into current military training exercises, and be able to cover unanticipated situations, so it does not interrupt the flow of training. Finally, all of the task requirements are set by existing military needs, and we were thus not able to tailor or simplify the domain to suit our purposes.

Simulated Tactical Air Combat

The application domain for TacAir-Soar is military training simulations. The United States military uses a wide range of tools to create simulated training environments. These include networked manned simulators for various vehicles (such as tanks, fighters, and bombers); digitized maps of world-wide terrain; software systems to simulate the dynamics of vehicles, weapons, and sensors; technology to link real soldiers and vehicles into a virtual environment; and software systems to simulate the results of interactions between various participants (human and automated) in the simulation.

The military uses simulation environments because they are cheaper, more flexible, and safer than live training maneuvers. Such training is particularly important in the post-cold war era, with general reductions in the defense budget. The tradeoff is that the military wants to "train as they fight". The concern is that some aspects of current simulation environments are not realistic enough to provide effective training.

Our task was to improve the behavior of individual, automated entities participating in simulation ex-

[0]Copyright (c) 1998, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

ercises. Before TacAir-Soar was developed, the state of the art for entity-level behavior used software systems called Semi-Automated Forces (SAFORs). SAFORs consist of high-fidelity models of vehicle dynamics (for example, using differential equations to model the control surfaces of an aircraft) together with relatively simple automated behaviors for controlling the vehicle, using finite-state machines. For example, a SAFOR aircraft can be told to circle a particular point at a certain altitude. Similarly, it can be told to shoot its air-to-air missiles at any airborne target that is detected within a specific "commit" range by its simulated radar. However, a human controller makes tactical decisions and uses a graphical tool to give the SAFOR new orders. The number of SAFORs that a single human can control depends on the type of platform being simulated, the expertise of the controller, the desired fidelity of the simulation, and other factors. In any event, a number of highly trained humans are required to oversee and adjust the runtime behavior of such entities. Aside from the expense of having humans in the loop, realism of the simulation suffers when these controllers must devote close attention to more than one entity at a time. There is a desire to increase the quality and autonomy of force behavior in order to increase realism and decrease the cost of training. To complicate matters further, there are additional goals to run as many forces on a single machine as possible, and to be able to run the simulation faster than real time for some applications.

In response, DARPA funded a project to create Intelligent Forces (IFORs), to use intelligent systems technology to develop autonomous systems that generate "human-like" behavior. Our group at the University of Michigan (plus researchers at the University of Southern California Information Sciences Institute, and Carnegie Mellon University) created a prototype system to generate behaviors for a small number of air-to-air missions (Tambe et al., 1995). The initial work combined the efforts of two university professors, two research scientists, and two programmers, with the research scientists working full time on the implementation of intelligent behaviors, and the programmers working full time on maintaining the software architecture, infrastructure, and interfaces. Two years into the project the groups divided, with the ISI group focusing on behaviors for missions flown by rotary-wing aircraft (Hill et al., 1997). Our group at UM hired an additional research scientist and programmer, and continued the work on fixed-wing aircraft with a team of five. This effort led to the development and deployment of TacAir-Soar. The currently deployed version of the system was completed in October, 1997.

The Simulation Environment

Before describing the details of the system that implements behavior for tactical combat simulations, it is worth discussing the details of the environment in which the system operates. We have already presented a general overview of military simulations. This section focuses on the details of the simulation structure with which TacAir-Soar currently interacts.

The core of the simulation environment is a network attached to a number of simulation systems (see Figure 1). Some systems are manned vehicle simulators (for tanks, aircraft, etc.). Some are real vehicles, equipped with hardware and software to interface with the simulation network. Finally, some are workstations running software to simulate various types of vehicles and human forces. STOW '97 did not include the participation of real vehicles or manned simulators, but they have been part of other tests and exercises in which TacAir-Soar was used.

In this simulation environment, each workstation maintains its own copy of fixed aspects of the environment (such as a detailed terrain database). Individual machines simulate the entities and changeable environmental features. In addition, each station predicts position information for other entities, reckoning from the last known entity state vector. Each workstation broadcasts new information on the network when it is simulating an entity that undergoes a change in state that cannot be predicted by the other machines (like a change in heading or velocity). Any other station on the network can use the new information to change its local knowledge of the entity, and then detect possible interactions between entities or with environmental features. Some of the distributed machines are devoted to special-purpose needs. For example, ordnance servers exclusively simulate the dynamics of guided weapons. Weather servers simulate meteorological features, sometimes using information from live feeds in the area of the world in which the simulation takes place.

TacAir-Soar runs within a simulation system called ModSAF (Calder et al., 1993). The ModSAF program provides realistic simulation of a large variety of military vehicles (including fixed-wing aircraft), weapons systems, and sensors. The system also provides a graphical interface for creating and controlling SAFORs, but this portion of ModSAF was not used by TacAir-Soar. Rather, we used additional code, called the Soar-ModSAF Interface (SMI; Schwamb, Koss, & Keirsey, 1994), to translate simulated sensor and vehicle information into the symbolic representation used by Soar and to translate Soar's action representation into ModSAF function calls (e.g., to control the vehicle and to manipulate weapons, sensors, and radios). We designed the SMI to provide a realistic interface to TacAir-Soar agents. That is, the SMI only passes to TacAir-Soar information that a real pilot would normally receive in a similar situation via cockpit readouts, the radar screen, visually, etc. Likewise, TacAir-Soar can only send commands to the interface that map onto the types of controls that real pilots have to fly the aircraft, employ weapons, manipulate sensors, and communicate. Each "instance" of TacAir-Soar controls a single entity.

We maintain a strong distinction between the code that generates intelligent behavior and the code that simulates physical systems. This distinction is not al-

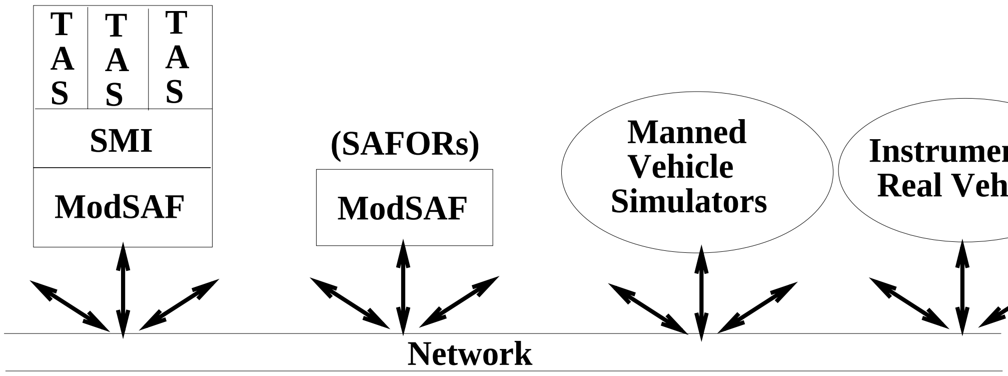

Figure 1: TacAir-Soar's niche in the simulation environment.

ways made in the development of SAFORs, but we found that it eased development, knowledge acquisition, and verification. This design will also ease the transition of TacAir-Soar to other simulation platforms. In the current deployment, ModSAF performs the simulation of the physical world; TacAir-Soar represents the mind that senses, reasons, and creates intentions; and the SMI represents an idealization of the perceptual-motor processes that allow the mind to interact with the world. We have also developed additional tools to translate orders from existing military tools into mission specifications for TacAir-Soar (Coulter & Laird, 1996; Kenny et al., 1998), and for commanders to update an entity's orders dynamically during execution (Jones, 1998). These tools allow us to present TacAir-Soar to the training audience as a "seamless" participant in the overall exercise. Laird et al. (1998) provide a more thorough discussion of this integration.

The TacAir-Soar System

TacAir-Soar is a rule-based system, built within the Soar architecture for cognition (Laird, Newell, & Rosenbloom, 1991). Soar uses production rules as the basic unit of long-term knowledge (or the "program"). Soar also provides architectural support for objects called "operators" and "goals". Each operator and goal can be viewed as a grouping of one or more production rules. TacAir-Soar currently contains approximately 5200 production rules, organized into about 450 operators and 130 goals. The rules represent the sum total of the system's knowledge about all of the missions it can fly. Each agent loads all of the rules, but does not usually use them all over the course of a single mission. There are significant numbers of general rules that are used in many different missions, and some rules that only apply in very specific situations.

Operators in Soar generally represent discrete, relatively fine-grained steps in a chain of reasoning. In the tactical air domain, some example operators are "Determine the identity of a radar blip", "Select a missile to fire", "Push the fire button", and "Set the aircraft to fly a particular heading". Unlike many other production system architectures, Soar requires a uniform representation for both selecting which operator to execute and implementing its functions. All production systems use the conditions of rules to determine candidate operators, but when there are multiple candidates many architectures use a fixed, implicit algorithm for selecting between the candidates, and then explicitly execute a list of operator actions. In contrast, Soar systems use rules to suggest choices between the candidate operators, and then use more rules to execute the set of actions for each operator. Thus, individual rules comprise a set of fairly simple conjunctive conditions (that are efficient to match) plus a set of discrete actions. However, these rules can fire in parallel and in sequence in order to implement complex conditional or iterative actions for each operator.

In our representation, explicit goals are a proper subset of operators. The Soar architecture generates a new goal structure when an operator takes more than a small, discrete unit of time (one *decision cycle*, in Soar parlance) to execute. The new goal provides a context for the system to apply more operators. For example, one of TacAir-Soar's operators is "Intercept an aircraft". This is not a discrete action that the system can "just do", so Soar creates a new goal structure. In the context of the new goal, the system proposes additional operators, such as "Achieve proximity to the aircraft", "Lock my radar on the aircraft", or "Employ weapons against the aircraft". The Soar architecture strongly supports hierarchical reasoning and execution, which is one of the reasons we used it. This support turned out to be important for a number of reasons. Hierarchical reasoning and execution directly address the requirements of the task.

The Advantages of Execution Hierarchies

One obvious advantage to a hierarchical knowledge representation is that the military structures nearly everything it does hierarchically. In particular, command and control hierarchies delegate different tasks to different levels of forces. Even small groups of 2 or 4 aircraft use a hierarchical command structure. One of TacAir-Soar's requirements is to interact with other forces within the command structure, so it is clear that the system must reason about and with hierarchies for at least part of its behavior. In the interests of a uniform integration, we extended the hierarchy into all of the reasoning that the system performs. In addition to interacting with others, this includes breaking personal goals into further decomposable subgoals and actions. Additionally, a hierarchical representation enhances the extendibility and maintainability of the system, by allowing us to compartmentalize knowledge, so it can be reused across appropriate contexts.

The hierarchical representation is also important to flexibly using default behaviors. When we began the TacAir-Soar project, our attitude was that we would not (initially) build a "complete" agent. The entire domain of tactical air combat is certainly huge, so we expected to spell out clearly which subset our system would address, and engineer the knowledge necessary to support those behaviors. While this initial assumption has held to some extent, it eventually became clear that we needed much broader coverage of behavior. The training audience (the military commanders and soldiers interacting with the simulation) found it unacceptable for a simulated entity not to do *something* in response to any situation, even if the behavior was not 100% correct, and even if our subject-matter experts had decided those situations were not worth our effort. This suggested a need for TacAir-Soar to include default behaviors, in order to perform reasonably in those situations for which we had not specifically engineered knowledge. The hierarchical representation of knowledge made it possible to create default actions for different levels of goals. Thus, rather than having only one or two brittle default behaviors, the defaults are sensitive to different types of situations and goals, so the system will generally do something reasonable.

As an example, when the system is trying to employ weapons against a target, there are many different tactics and flight patterns to use, based on the current situation. However, a generally good default action is to fly a collision course to the target, in order to approach it as rapidly as possible. In contrast, once the system has maneuvered the target into weapons range, its next action (a subgoal of employing weapons) is to fire a missile at the target. Once again, there are certain good actions to take in order to get off the best shot possible. But if something goes wrong, in this case a good default action is to point straight at the target (rather than flying a collision course). Thus, the appropriate default behaviors are dependent on the current reasoning context imposed by the knowledge hierarchy.

Opportunistic Operators

At execution time, the goal hierarchy generates top down. Most of the time every TacAir-Soar agent has one general active goal, called `Execute-Mission`. Depending on the mission type and the current situation, the agent will select one of a number of subgoals, including `Fly-Route`, `Intercept-Target`, or `Follow-Leader`. These in turn will have subgoals. Depending on the situation, the system creates more specific goals until it reaches a level where it can manipulate the environment directly, such as to issue a command to fly the aircraft to a new heading or altitude.

There are also a number of important actions that do not fit nicely into a strict goal hierarchy. These include relatively context-insensitive actions, such as "Determine threat of unknown contacts", and actions in the service of sweeping, general goals, such as "Evade a missile", which serves the general goal of survival. Operators for these actions are not part of the explicit goal hierarchy. Rather, they are represented as *opportunistic* operators, which can be selected independently of the currently active, explicit goals. This mixture of goal-driven and opportunistic behaviors is crucial to capturing the ways in which humans interleave complex sequences of actions in the service of multiple goals (Jones et al., 1994).

Situation-Representation Hierarchy

These two classes of operators play key roles in generating the rich variety of behaviors required by the task. In each case, operators and goals activate at execution time based on the current *situation*. This raises the question of how that situation is represented. Certainly, one important part of the agent's situation is the current set of active goals. A great deal of the situation is also provided by symbols describing the current state of the external environment, as seen through the vehicle's sensors. This is where a second representational hierarchy comes into play.

There are usually a number of different ways to describe an air combat situation, based on the different types of decisions that need to be made. Consider the case where an aircraft has a potential target contact on its radar screen. In some cases, it is important to reason about the heading or bearing of the contact. In others, it is important to notice whether the target is flying to the left or the right (which can be computed from a combination of the target's heading and the agent's heading). It is also often important to reason about the target's position relative to other contacts that may be flying with the target. In general, it is difficult to restrict the agent to reasoning about the primitive attributes that describe the situation. It is preferable to combine these attributes based on the current context, in order to make different types of decisions.

Thus, many of TacAir-Soar's rules are not associated directly with an operator. They are instead part of a data-driven, situation-interpretation hierarchy, which generates descriptions of the situation from the bottom up. The benefits of the interaction between the top-down and bottom-up hierarchies are illustrated by the example shown in Figures 2 and 3. Assume Figure 2 describes an agent's current "mental state" at some point in time. The boxes represent a subset of the agent's active goals, and the other items are a subset of the

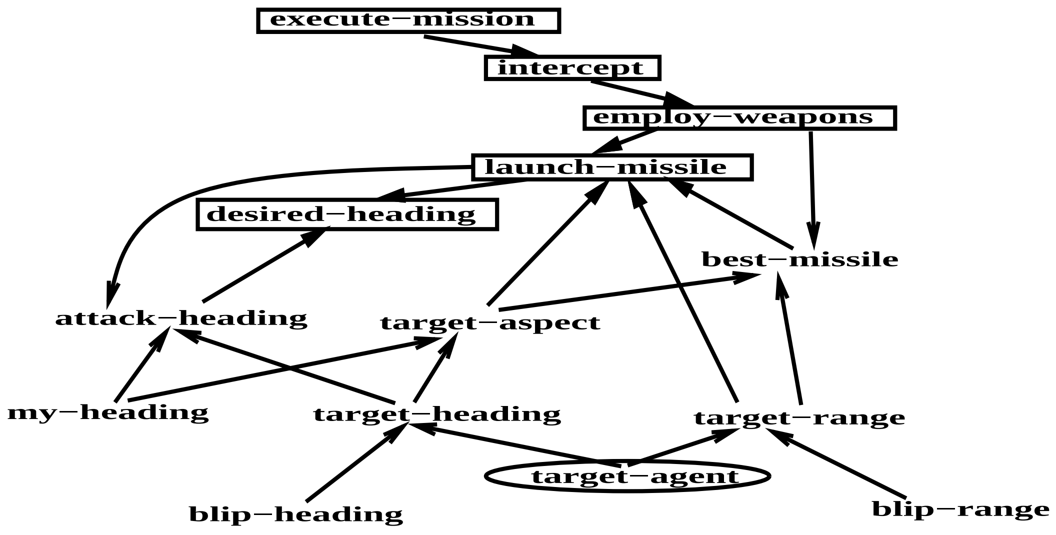

Figure 2: A sample active goal hierarchy for TacAir-Soar.

current situation description. The generation of these goals and features is fairly rapid, but it does take some time. Thus, it is undesirable for the agent routinely to expend significant effort maintaining the hierarchies, because it may run out of time to react.

However, because of the hierarchical representation, the system can make incremental adjustments to its current mental state as the situation warrants. In Figure 2, the agent has chosen a missile to employ against a target, and is using the target's position information to aim the aircraft for the best shot. If the target turns a small amount, it may not lead to any change in the active hierarchies at all. Rather, the change in target heading will cause a rule to fire that computes a new attack heading, and the agent will adjust the aircraft heading in response. If the target turns by a larger amount, the new geometry of the situation may take the target out of the envelope of the currently selected missile. This could cause the agent to delete its current `Launch-Missile` goal, select a new missile to use, and then create a new `Launch-Missile` goal. Finally, if the target turns by an even larger amount (for example, to run away), the agent may have to make significant adjustments, leading to the situation in Figure 3. In this case, the agent determines that the target may be running away, so there is no longer a need to shoot it. Therefore, the agent deactivates the goal to launch a missile against the target, and instead pursue's the target, to observe its behavior for some time.

This example illustrates TacAir-Soar's solution to the combined constraints of generating complex behavior in an efficient manner. The system's internal representation is extremely intricate at any point in time (the figures are highly simplified). However, the interaction of top-down and bottom-up hierarchies, together with opportunistic operators, allows a smooth and efficient implementation of reactive, yet goal-driven, behavior.

The Problem of Efficiency

The representations above provide a framework for creating complex reactive behavior, but we needed further work to address the real-time constraints on TacAir-Soar. The system needs enough time to make appropriate, human-like decisions, even when the environment is changing quickly. Additionally, simulation must be as inexpensive as possible, while meeting the training goals. One way to keeps costs down is to execute as many automated forces on a single computer as possible. Thus, there are multiple reasons for TacAir-Soar to be efficient. We increased the efficiency of the system in a number of ways, which can be categorized as enhancing the rule-based architecture, moving non-symbolic computation into the SMI, improving the context sensitivity of processing, and using fast but cheap hardware.

The first point to mention is the reïmplementation of the Soar architecture. In the Summer of 1992, just prior to the start of this project, Doorenbos (1993) completed a new version of Soar, which is written in C (previous versions were in Lisp) and contains a very efficient implementation of the RETE rule-matching algorithm (Forgy, 1982). The new version of Soar was approximately 15-20 times faster than the previous version. This system provided the required infrastructure for us to even hope to build systems for intelligent, real-time control. It also gave us the speed to run our agents faster than real time in some situations, and to run multiple agents on a single machine.

As the project progressed, we discovered other architectural adjustments that address efficiency. One adjustment arose from the fact that TacAir-Soar does not currently make any use of the Soar architecture's built-in learning mechanism. This is because the current system models expert behavior, not the acquisition of expert behavior. The learning mechanism has some over-

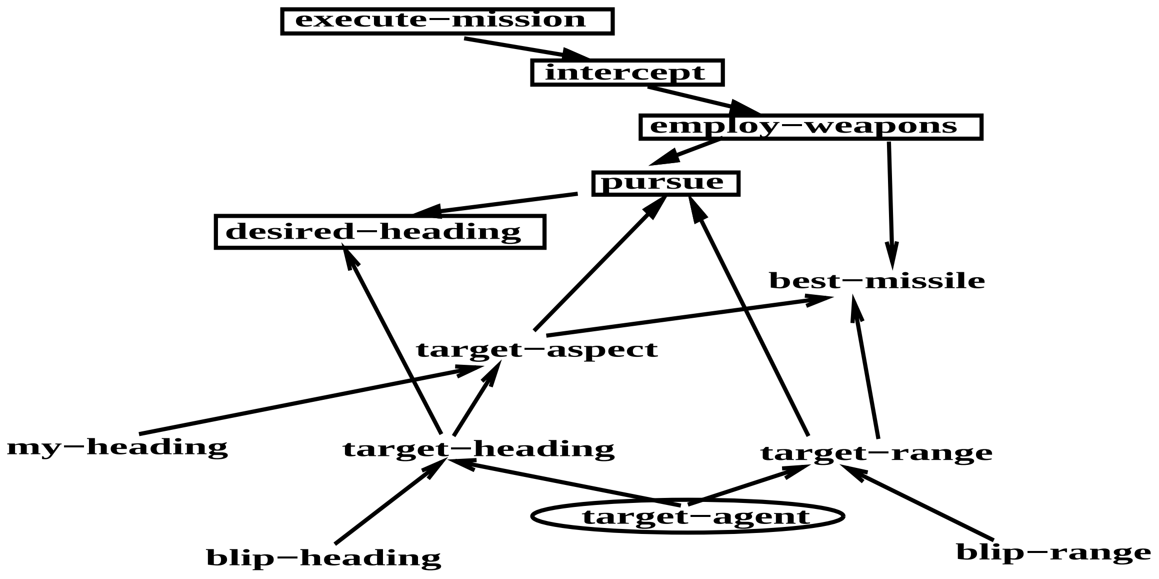

Figure 3: TacAir-Soar changes the goal hierarchy in response to changes in the situation-representation hierarchy (and vice versa).

head associated with it, so we selectively disabled the support for learning. We should stress that this does not represent a change in the "Soar unified theory of cognition" (Newell, 1990), but is a departure based on the pragmatic goal of increasing efficiency when learning is not important. However, it is worth noting that our experiences with TacAir-Soar have suggested some *potential* adjustments to the theory, which are being explored (Wray, Laird, & Jones, 1996).

Another technique to increase efficiency was to migrate processing from rule-based behaviors into the SMI. Initial implementations restricted the agent's sensory input from the SMI to sparse representations. For example, for an active radar contact, the SMI would pass only those numbers that are accessible on a real radar display. To perform intercepts, however, a pilot uses geometric and spatial reasoning, such as to compute a collision course. Early versions of TacAir-Soar contained rules to compute such values. The rules fired for a new computation every time one of the relevant input values changed. Combat aircraft often fly near the speed of sound, and they maneuver against each other, so these situation-interpretation rules can fire very often. However, such fixed, goal-independent processing is much more efficiently implemented in C code than in the match-and-fire interpretation cycle inherent in rules. We recovered significant processing time by moving such expensive, "non-intelligent" computations into the SMI. The new version of the SMI passes a contact's geometric information (such as the collision course) along with the normal numeric information. TacAir-Soar uses this information as before, without the inefficient computational overhead.

Another rather naïve (in hindsight) early design decision allowed the system to process more information than is strictly necessary for decision making. For example, early versions of the situation-interpretation hierarchy processed primitive input information from the SMI into as many different representations as it could. Also, the SMI would compute many different geometric quantities for every active radar contact. The motivation behind this approach was to make the system more easily extendible. The "right" set of features would be available any time we added new behaviors, because the system would represent the situation in so many different ways. As a simple example, based on numeric sensor information for a radar contact, we would compute whether the contact is to our left or right, in front of us or in back of us, pointing at us, turning, descending or climbing, and so on.

This approach *did* make it easier to expand the capabilities of the system. However, we soon disovered that TacAir-Soar was doing so much processing, it was sometimes running out of time to make appropriate decisions. Thus, we tempered our goals of software extendibility in the service of a more efficient knowledge representation. To this end, we only use situation-interpretation rules to compute new features if those features are relevant to the agent's current context. Similarly, the SMI does not need to compute *all* the possible geometric information for most contacts. If the agent knows that a radar contact on its scope is friendly, there is no reason to compute relational position features, because friendly forces can often be ignored. Similarly, there may be no reason to compute whether an enemy aircraft is pointing at the agent's aircraft if the enemy aircraft is known not to carry air-to-air missiles. In response, we equipped the system with methods to focus attention and interpret the situation based on the current context. These changes were sometimes tedious, but the purely functional concern of efficiency forced us to create a more plausible representation of human reasoning.

As a final point on efficiency, we should not ignore the power of brute-force solutions. Over the five years that TacAir-Soar has been developed, computer system speeds have continued their familiar dramatic improvements. For STOW '97, we did not try to find the fastest computers available, because low cost was also a goal. We did try to find the best ratio of cycle time to system cost, and ended up using Intel-based machines running the Linux operating system. With these machines, and our attention to efficiency within TacAir-Soar's design, we were able to run up to around 6 instances of TacAir-Soar simultaneously on a single machine, without significant degradation in the quality of behavior.

Details of Deployment

TacAir-Soar has taken part in a number of tests, technology demonstrations, and operational exercises (involving real training of military personnel). The system is fulfilling its role, providing an important part of the environment for simulation training. TacAir-Soar's is primarily used at the WISSARD Laboratory on the Oceana Naval Air Station, in Virginia Beach. In early tests, the system participated in simple engagements (e.g., 1 vs. 1 air-to-air combat), fighting against humans who were "flying" various types of flight simulators. This type of test fit into the original role we had envisioned for TacAir-Soar: providing intelligent automated opponents to train small groups of combat pilots in offensive and defensive tactics. However, there are many different potential applications of this technology, and the particular requirements for this system changed as the project progressed.

In STOW '97, TacAir-Soar was used for commander training, where the trainees receive reports from the battlefield, and task combat units to perform various functions. The simulation runs concurrently, leading to new battlefield reports and an assessment of the effectiveness of the orders. Various types of simulators play out these training scenarios, including entity-level SAFOR simulations and aggregate-level "war game" simulations. The military uses TacAir-Soar to increase the realism of such simulations. In addition to generating more realistic general behavior, TacAir-Soar communicates (in simplified English) and coördinates with humans. Commanders actually observe and interact with the forces as the simulation plays out, leading to a much more realistic training scenario. TacAir-Soar is primarily used for this type of theater-level command-and-control training. However, there are plans to make more frequent use of TacAir-Soar to train small groups of pilots in adversarial and cooperative situations. A number of active and retired military personnel have expressed surprise and enthusiasm at the difference between TacAir-Soar's behavior and other simulations.

As might be expected, expanded training requirements greatly increased the demands on the project. The ultimate requirements were for the system to include appropriate behaviors for every mission that is performed on a fixed-wing aircraft in the United States military, as well as to provide behaviors for "opponent force" (OPFOR) aircraft and aircraft from the United Kingdom's Royal Air Force. The missions that TacAir-Soar flies include offensive and defensive counter-air, close-air support, suppression of enemy air defense (SEAD), strategic attack, escort, reconnaissance/intelligence, airborne early warning, mid-air tanking, and forward air control. In addition, the system had to be resistant to generating anomalous behavior or software failures, so that it would not interrupt the continuity of the training exercise. Finally, the system had to integrate seamlessly with existing military structure for exercises. Standard procedure in the United States military is to generate an Air Tasking Order (ATO) every 24 hours. This order specifies airborne mission information for the next day, created and distributed with a database tool. For live exercises, the ATO is sent to wing commanders, who then create specific orders for the details of each aircraft flight. We developed additional tools for scenario management (Coulter & Laird, 1996; Kenny et al., 1998), in order to automate this process. These tools translate the ATO into mission specifications for TacAir-Soar, so the interactions between commanders and our system are as similar as possible to their interactions with real wing commanders and combatants.

TacAir-Soar's participation in STOW '97 took place over 48 straight hours. In that time, the system generated behavior for 722 scheduled flights, including every type of mission that TacAir-Soar is capable of flying. There were up to 4000 total entities in the simulation environment at a time, with TacAir-Soar controlling between 30 and 100 at a time on 28 different machines. Missions varied in length from 90 minutes to 8 hours. After the missions had been entered into the computer, the behaviors generated by TacAir-Soar were entirely autonomous. The entities were sometimes re-tasked by active-duty personnel, but this occurred within acceptably realistic interactions, using a speech recognition system or a graphical tool to send the entities simulated radio messages in semi-natural language. There were generally one or two people monitoring all of the entities to keep an eye out for incorrect behavior. Unfortunately, we do not have cost-estimate comparisons, but this is significantly less manpower than is required to run a similar number of SAFOR aircraft.

Future Plans

STOW '97 was a complete technical success. Based on this success, the system continues to be used for technology demonstrations at the Oceana Naval Air Station, and there are plans to use the system in additional operational training exercises in the coming year. Under the auspices of a new company, Soar Technology, we have installed the system at additional military training sites, and we are continuing its development in a number of ways. One obvious area of development is to expand and broaden the general behaviors and missions that TacAir-Soar performs, to increase its participation and realism in theater-level simulations like STOW '97. To address other types of training, we are also putting significant effort into engineering TacAir-Soar for specific, individual training roles. Some example roles are synthetic wingmen to provide partners to individual lead fighter pilots, synthetic fighters to interact closely with human AWACS controllers, and high-fidelity enemy aircraft to train fighter pilots in air-to-air engagements. These focused behavior areas require us to emphasize the development of TacAir-Soar's ability to communicate using natural language and speech, and to engineer deeper knowledge of the targeted mission areas. Additional development involves parameterizing the behavior model so it more realistically emulates the differences in behavior between different types of pilots, and pilots in different branches of the military (each with their own standard operating procedures).

In the academic arena, we are using TacAir-Soar as the basis for a variety of research projects in intelligent systems and cognitive science. These projects includes studies of how intelligent systems can use explanation-based learning in situated domains (Wray, Laird, & Jones, 1996), how systems can acquire expert-level combat pilot knowledge through training (van Lent & Laird, 1998), and how we can use TacAir-Soar to make predictions about the effects of fatigue on pilot behavior (Jones, Neville, & Laird, 1998).

Acknowledgements

Milind Tambe and Paul Rosenbloom participated in the design and implementation of early versions of TacAir-Soar, before breaking off to work on the system for rotary-wing vehicles. Karl Schwamb, Frank Koss, Karen Coulter, and Patrick Kenny provided programming support in the form of the SMI and other software tools. Randall Hill, Jonathan Gratch, and Johnny Chen also aided in the development of the rotary-wing system, and their work provided us with valuable insights. The version of TacAir-Soar described in this paper was developed under the auspices of the University of Michigan and the Information Sciences Institute of the University of Southern California, under contract N00014-92-K-2015 from the Advanced Systems Technology Office of DARPA and ONR, and contract N6600I-95-C-6013 from the Advanced Systems Technology Office of DARPA and the RDT&E division of NOSC.

References

Calder, R., Smith, J., Courtenmanche, A., Mar, J., & Ceranowicz, A. 1993. ModSAF Behavior Simulation and Control. *Proc. of the 3rd Conf. on Computer Generated Forces and Beh. Representation.* Orlando, FL.

Coulter, K. J., & Laird, J. E. 1996. A Briefing-Based Graphical Interface for Exercise Specification. In *Proc. of the 6th Conf. on Computer Generated Forces and Beh. Representation*, 113–117. Orlando, FL.

Doorenbos, R. 1993. Matching 100,000 Learned Rules. In *Proc. of the 11th Nat. Conf. on Artificial Intelligence*, 290–296. Menlo Park, CA: AAAI Press.

Forgy, C. L. 1982. Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–38.

Hill, R. W., Chen, J., Gratch, J., Rosenbloom, P., & Tambe, M. 1997. Intelligent Agents for the Synthetic Battlefield: A Company of Rotary Wing Aircraft. In *Proc. of 9th Conf. on Inn. App. of Artificial Intelligence*, 1006–1012. Menlo Park, CA: AAAI Press.

Jones, R. M. 1998. A Graphical User Interface for Human Control of Intelligent Synthetic Forces. In *Proc. of the 7th Conf. on Computer Generated Forces and Beh. Representation.* Orlando, FL.

Jones, R. M., Neville, K., & Laird, J. E. 1998. Modeling Pilot Fatigue with a Synthetic Behavior Model. In *Proc. of the 7th Conf. on Computer Generated Forces and Beh. Representation.* Orlando, FL.

Jones, R. M., Laird, J. E., Tambe, M., & Rosenbloom, P. S. 1994. Generating Behavior in Response to Interacting Goals. In *Proc. of the 4th Conf. on Computer Generated Forces and Beh. Representation*, 317–324. Orlando, FL.

Kenny, P. G., Coulter, K. J, Laird, J. E., & Bixler, S. 1998. *Integrating Real-World Mission Specification and Reporting Activities with Computer Generated Forces at STOW-97.* In *Proc. of the 7th Conf. on Computer Generated Forces and Beh. Rep.* Orlando, FL.

Laird, J. E., Coulter, K. J., Jones, R. M., Kenny, P. G., Koss, F. V., & Nielsen, P. E. 1998. Integrating Intelligent Computer Generated Forces in Distributed Simulation: TacAir-Soar in STOW-97. In *Proc. of the 1998 Simulaton Interop. Workshop.* Orlando, FL.

Laird, J. E., Newell, A., & Rosenbloom, P. S. 1991. Soar: An Architecture for General Intelligence. *Artificial Intelligence*, 47:289–325.

Newell, A. 1990. *Unified Theories of Cognition.* Cambridge, MA: Harvard University Press.

Schwamb, K. B., Koss, F. V., & Keirsey, D. 1994. Working with ModSAF: Interfaces for Programs and Users. In *Proc. of the 4th Conf. on Computer Generated Forces and Beh. Representation.* Orlando, FL.

Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S., & Schwamb, K. B. 1995. Intelligent Agents for Interactive Simulation Environments. *AI Magazine*, 16(1):15–39.

van Lent, M., & Laird, J. E. 1998. Learning by Observation in a Complex Domain. In *Proc. of the Workshop on Knowledge Acquisition, Modeling, and Management.* Banff, Alberta.

Wray, R. E., III, Laird, J. E., & Jones, R. M. 1996. Compilation of Non-Contemporaneous Constraints. In *Proc. of the Thirteenth Nat. Conf. on Artificial Intelligence*, 771–778. Menlo Park, CA: AAAI Press.