From **AI Magazine** *Spring 1999*
Reprinted by permission

Automated Intelligent Pilots for Combat Flight Simulation*

Randolph M. Jones[1], John E. Laird[2], Paul E. Nielsen[3],
Karen J. Coulter[2], Patrick Kenny[3], and Frank V. Koss[3]

[1]Colby College
5847 Mayflower Hill Drive
Waterville, ME 04901-8858

[2]University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109-2110

[3]Soar Technology, Inc,
3600 Green Court, Suite 480
Ann Arbor, MI 48105

rjones@colby.edu, laird@umich.edu, nielsen@soartech.com,
kcoulter@umich.edu, pkenny@soartech.com, koss@soartech.com

Abstract

TacAir-Soar is an intelligent, rule-based system that generates believable "human-like" behavior for large scale, distributed 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 real-time hierarchical execution of complex goals and plans, communication and coordination with humans and simulated entities, maintenance of situational awareness, and the ability to accept and respond to new orders while in flight. The system is currently deployed at the Oceana Naval Air Station WISSARD Lab and the Air Force Research Laboratory in Mesa, AZ. Its most dramatic use was in the Synthetic Theater of War 1997, which was an operational training exercise that ran for 48 continuous hours during which TacAir-Soar flew all U.S. fixed-wing aircraft.

Keywords: intelligent agents, autonomous agents, simulation, real-time rule-based systems, hierarchical knowledge, Soar

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 was 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 facility at the Oceana Naval Air Station, and the Air Force Research Laboratory at Mesa, Arizona. TacAir-Soar consists of over 5,200 rules that are executed by the Soar architecture (Laird, Newell, & Rosenbloom, 1991). As TacAir-Soar has grown, it has participated in a number of tests, technology demonstrations, and operational training exercises. Its most dramatic use was in the Synthetic Theater Of War 1997 (STOW '97), held October 29-31, 1997 (Ceranowicz, 1998; Laird et al., 1998a; Laird et al., 1998b). STOW '97 was a Department of Defense Advanced Concept Technology Demonstration (ACTD) that was integrated with the United Endeavor 98-1 training exercise. As an ACTD, the overall goal of STOW '97 was to permit an early and inexpensive evaluation of advanced technologies that show promise for improving military effectiveness. The goal of our participation in STOW '97 was to demonstrate the ability to generate autonomous, real-time, high fidelity behavior for a large-scale simulation of a complete theater battle. STOW '97 included 722 individual sorties to be flown by U.S.

* All authors were affiliated with the University of Michigan during development of the project reported in this article.

fixed-wing aircraft. All of these sorties were assigned to TacAir-Soar agents, and the agents successfully flew over 95% of them. In July 1998, TacAir-Soar participated in the RoadRunner '98 training exercise where it flew both with and against human pilots "flying" in simulators.

The most obvious innovation of TacAir-Soar lies in the integration of a number of capabilities that support autonomous, intelligent behavior in a real-time, complex domain. TacAir-Soar generates appropriate behavior for *every* fixed-wing mission routinely used by the U.S. military in full-scale simulated exercises. A secondary, but equally challenging innovation is that TacAir-Soar was integrated into the military operational organizational structure so that existing methods and systems could be used for specifying missions, interacting with the aircraft in flight during mission execution, and post-mission reporting. Thus, the integration of capabilities within an intelligent system into an existing organizational structure was the combined challenge and theme of our work.

This paper starts with a review of the origins of this project and the requirements that shaped the development of TacAir-Soar. We then describe the simulation environment and TacAir-Soar's integration. The main body of the paper presents the structure of TacAir-Soar, relating specific design decisions back to the original requirements. This is followed by discussions of the deployment and payoffs of TacAir-Soar, as well as our ongoing efforts.

Historical Background

Our task was to improve the behavior of individual, automated entities participating in large-scale, entity-based simulation exercises. The military uses simulation extensively for training, analysis, and acquisition. Simulation is often cheaper, more flexible, and safer than alternatives, such as live training maneuvers. However, simulation is only an approximation of the real world, and may be ineffective if it deviates significantly from reality. Many of the early simulations used by the military represented only aggregate groups of vehicles and could not simulate the interactions between individual vehicles, sensors, and weapons systems. Instead, such systems use probability tables to determine the outcome of engagements between large forces. In contrast, entity-based simulations involve models of the dynamics of individual vehicles, their sensors, and weapons. This increases the realism of the simulation and allows more direct participation by humans. However, the added realism has the additional cost of requiring either people or automated systems to control all of the vehicles. For large-scale simulations where there are thousands of entities, this can be either extremely costly (using humans), or technically challenging.

Before we developed TacAir-Soar, the state of the art for automated entity-level behavior were 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 simple automated behaviors for controlling the vehicle, using hierarchical 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 must make the tactical decisions and use a graphical computer interface to give the SAFOR new orders. The number of SAFORs that a single human can control depends on many factors, including the type of platform being simulated, the expertise of the controller, and the desired fidelity of the behaviors. 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. Therefore, it was desirable to increase the quality and autonomy of force behavior in order to increase realism and decrease the cost of training. Complicating matters further, there were additional goals to run as many forces on a single machine as possible, and to run the simulation faster than real time for some applications.

To address these issues, DARPA funded a project to develop autonomous computer generated forces that generate "human-like" behavior. Our group at the University of Michigan, together with researchers at the University of Southern California Information Sciences Institute created a prototype system to generate behaviors for a small number of air-to-air missions (Tambe et al., 1995). This prototype laid out the basic design principles we followed throughout the project, and our earlier paper provides details on the design that are not covered in this paper.

The initial work was led by Professors John Laird (UM) and Paul Rosenbloom (USC/ISI) and also included two research scientists (Randolph Jones, Milind Tambe), and two research programmers (Frank Koss, Karl Schwamb). This original work included development of intelligent behaviors, and interfaces to the simulation environment, as well as maintaining the software architecture and infrastructure. Two years into the project, the group divided, with the USC/ISI group focusing on behaviors for missions flown by helicopters (Hill et al., 1997). Our group at Michigan continued with fixed wing aircraft, expanding the types of aircraft and missions, as well as creating software to support

the installation and use of the system in military training operations. An additional research scientist (Paul Nielsen) and two research programmers (Karen Coulter and Patrick Kenny) were hired at the University of Michigan. This effort led to the development and deployment of TacAir-Soar. The currently deployed version of the system was complete by October 1997, in time for the STOW '97 exercise.

The Simulation Environment

TacAir-Soar is embedded within real-time large-scale simulations of a battlefield. Time in the simulation corresponds to time in the real world, so both humans and computer forces must react in real time. The scale of the simulation is that of a theater of war. STOW '97 took place over 500 x 775 square kilometers with up to 3,700 computer-generated vehicles comprising all U.S. military services as well as opposition forces. The terrain was detailed, including 13,500 buildings, and over 20,000 destructible objects, runways, roads, bridges, bushes, rivers, bodies of water, etc. Some aspects of the terrain were dynamic, such as destructible buildings and bridges. Further, there were weather servers that simulated meteorological features such as clouds, rain, tides, and daylight, sometimes using live feeds from the area of the world in which the simulation took place.These simulations are completely distributed, with no central server responsible for maintaining all relevant information. STOW '97 used over 300 networked computers at six sites in the U.S. and England. All of the systems participating in the simulation, be they manned simulators, computer generated forces, simulations of complex munitions such as missiles, or even specially instrumented real vehicles, are responsible for maintaining their own model of the simulation environment. To minimize communication across the network, each simulation system maintains its own copy of the terrain and uses dead reckoning to predict the state of all entities that are relevant to it. Whenever an entity deviates from its projected position, it sends out data to the systems that are monitoring its behavior.

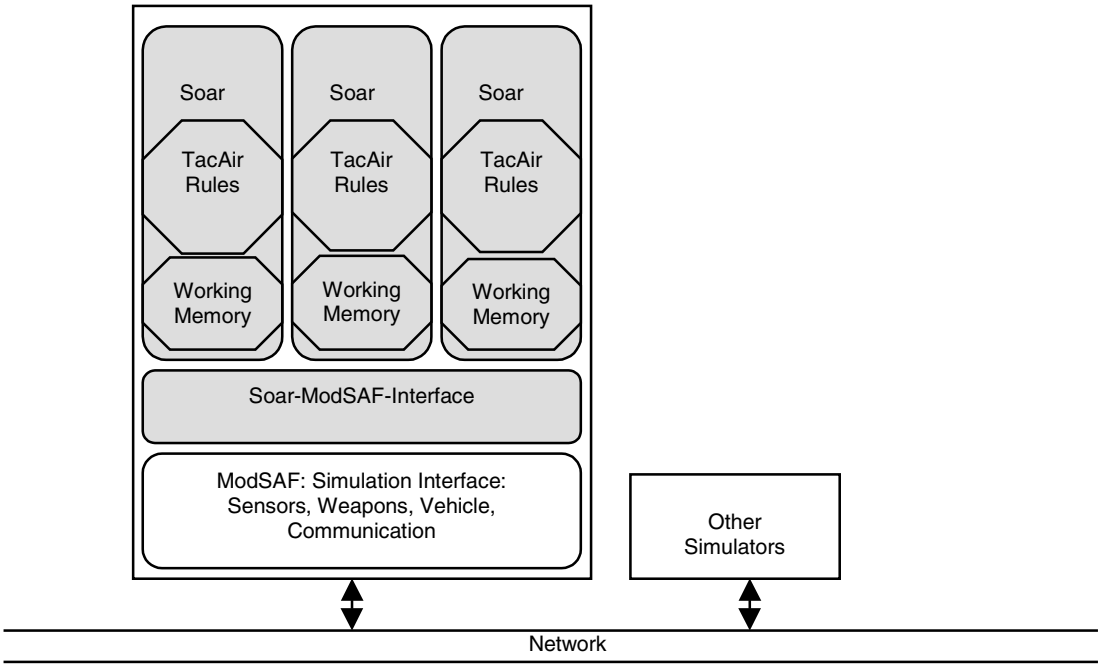

Figure 1: System organization for TacAir-Soar

As shown in Figure 1, TacAir-Soar participates in the simulation environment by interfacing with ModSAF (Calder et al., 1993). ModSAF includes all of the necessary software for interacting over the network with other entities in the simulation, as well as simulations of the vehicle dynamics, weapons, and sensors. TacAir-Soar connects to ModSAF via the Soar-ModSAF Interface (SMI; Schwamb, Koss, & Keirsey, 1994), which translates simulated sensor and vehicle information into the symbolic representation used by Soar and translates Soar's action representation into ModSAF function calls (e.g., to control the vehicle and to manipulate weapons, sensors, and radios). The SMI attempts to organize data so that TacAir-Soar has available the same information a pilot would have, including data from radar and vision sensors, as well as data on the status of the aircraft and messages received via its radios. This information is

added directly to the working memory of each entity and updated each cycle of the simulation. For a single entity, there are over 200 inputs and over 30 different types of outputs for controlling the plane. Each entity only has access to information from its own sensors, so that there is no possibility of "cheating".

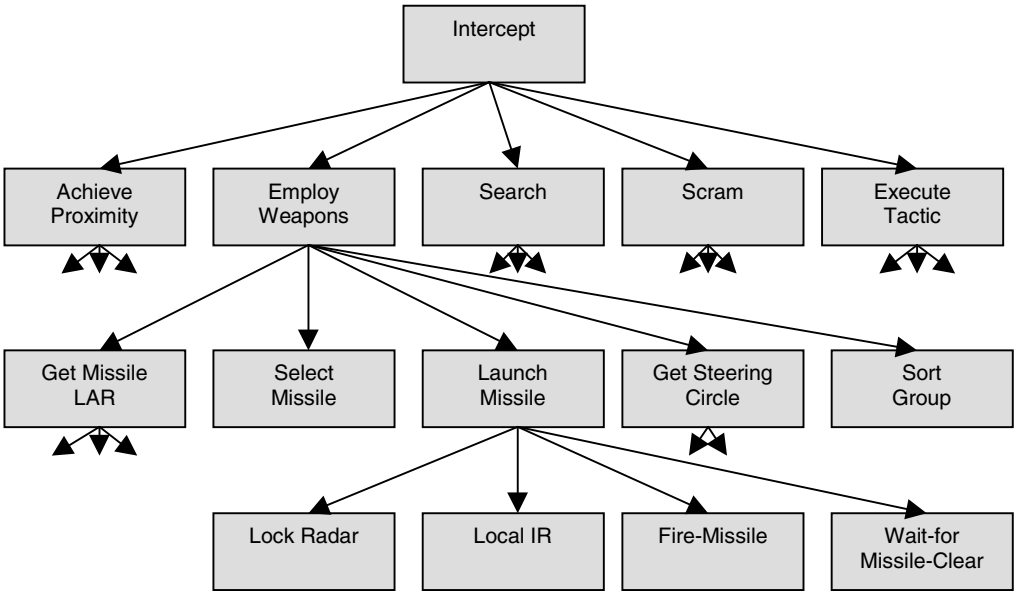

Figure 2: Example operator hierarchy decomposition

Each "instance" of TacAir-Soar controls a single entity. In a typical configuration, several TacAir-Soar entities run with ModSAF as a single process under Linux on a Pentium class machine with 256 Mbytes of memory. The large memory is required because of the size of the terrain database. A round-robin scheduler provides service to the TacAir-Soar entities and ModSAF. Approximately 10 TacAir-Soar entities run on a single 400MHz workstation with each entity executing 3-4 times a second, which is sufficient to generate realistic behavior. During STOW '97, we ran up to 102 TacAir-Soar entities at a time, distributed across 28 Pentium Pro machines.

The TacAir-Soar System

TacAir-Soar is a symbol-processing, 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"). Rules in Soar are used to propose, select and apply "operators", which correspond to the actions and goals human pilots perform during a mission. Abstract operators, such as "intercept an enemy", act as goals, which are dynamically decomposed by rules proposing more primitive operators to achieve the abstract actions. These rules test the current situation, including the available sensors and mission parameters, avoiding fixed, scripted responses. For example, as shown if Figure 2, 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", "Search for the aircraft", or "Employ weapons against the aircraft". When an abstract operator has been fully decomposed so that a primitive operator is selected, rules create appropriate commands for the motor system such as "Select a missile to fire", "Push the fire button", and "Set the aircraft to fly a particular heading." Thus, Soar supports dynamic hierarchical decomposition and execution of complex operators. This organization appears to correspond to the way human pilots organize their knowledge and provides readable traces of behavior (in terms of operator selections and decompositions).

Although dynamic hierarchical decomposition is important for encoding structured knowledge such as in official doctrine and tactics, it can be overly restrictive when there are multiple goals that must be pursued. In TacAir-Soar, the operator hierarchy usually corresponds to the explicit goals and subgoals of the mission being executed. However, there are additional, implicit goals in TacAir-Soar that can give rise to actions in the world. For example, during a flight, a plane may notice a new blip on its radar. Bringing together different information sources and interpreting the identity of the blip is a deliberate act performed by an operator. That operator is not directly part of a goal of executing

a mission, but is part of an implicit goal of maintaining situational awareness. Similarly, a plane may need to take evasive actions when it is in danger, in response to a survival goal. These operators 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).

Both goal-driven and opportunistic operators are proposed based on the current *situation*. The operator hierarchy defines only a part of that situation, which also includes sensory data and an internal model of the other agents in the world. These representations are all created in support of making decisions, and we have found that it is useful to organize the internal models hierarchically in a manner analogous to decisions that need to be made in the operator hierarchy.

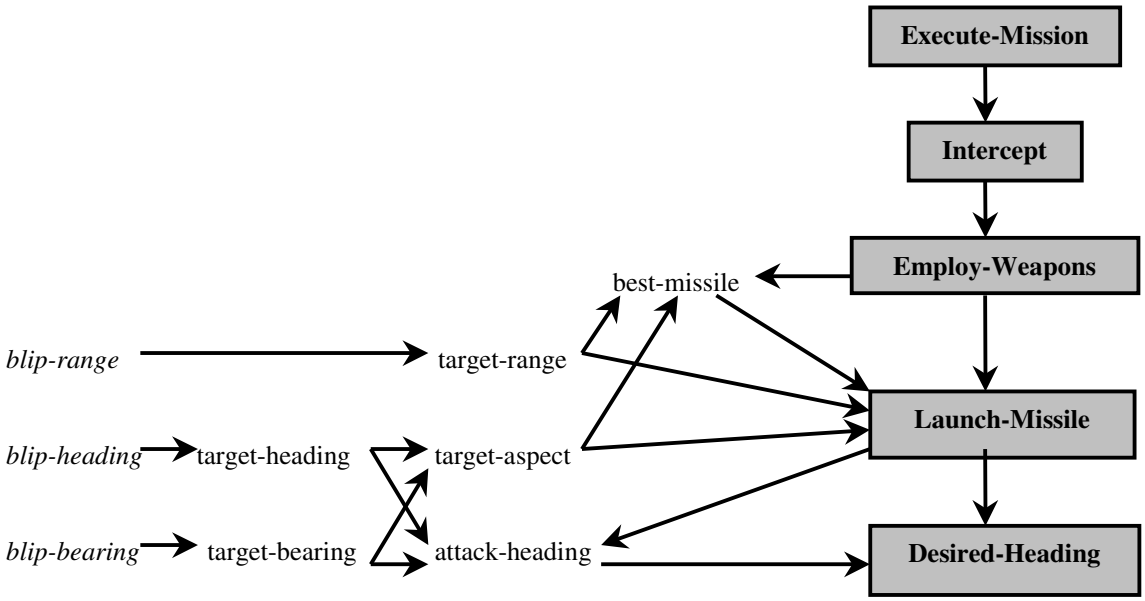

Figure 3: Sample active goal and situation hierarchies

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, in order to make different types of decisions. These hierarchical representations are built by TacAir-Soar rules that are not directly associated 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 3 and 4. Figure 3 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 current situation description. The generation of these goals and features is fairly rapid (Soar is capable of firing hundreds of rules per second), but it does take some time and computational resources. Thus, it is undesirable for the agent routinely to expend significant effort maintaining the hierarchies. However, because of the hierarchical representation, the system can make incremental adjustments to its current mental state as the situation warrants. In Figure 3, 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 4. 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 pursues 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.

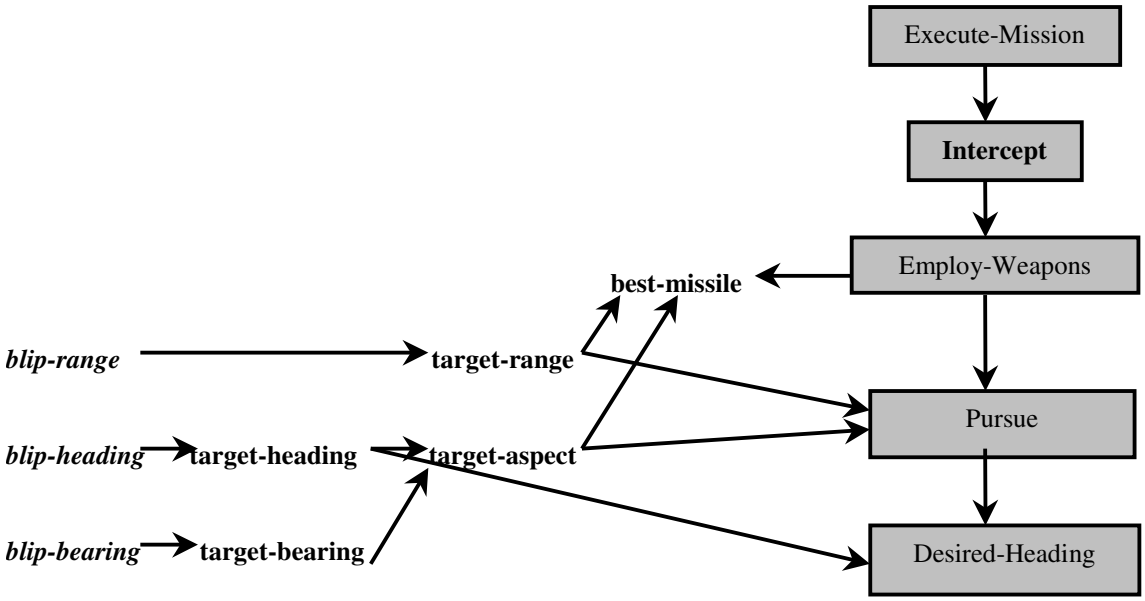

Figure 4: TacAir-Soar changes the goal hierarchy in response to changes in the situation-representation hierarchy (and vice versa)

Behavioral Requirements

In this section, we review the requirements that were set for TacAir-Soar and how we attempted to achieve them.

1. Fly all types of missions using appropriate doctrine and tactics.
2. Fly many different types of aircraft.
3. Generate human-like behavior.
4. Coordinate behavior with other entities.
5. Behave with low computational expense.

Fly all types of missions using appropriate doctrine and tactics.

Our most important requirement is to populate the battlespace with entities that can fly all of the missions commonly flown by U.S. Navy, Marine, and Air Force planes. For STOW '97 there were over ten different types of missions covering defensive and offensive air-to-air, air-to-ground missions, support (such as refueling), intelligence gathering, and command and control. To encode the knowledge for these missions we read manuals and interviewed subject matter experts (SMEs). This is an iterative process, where we interview experts, build systems that perform the missions, demonstrate them to experts, receive feedback, and revise the behaviors as appropriate.

There is a significant amount of knowledge required to fly all of the missions according to appropriate doctrine. However, by encoding the knowledge hierarchically, TacAir-Soar shares knowledge across many different missions and tactics. For example, air-to-air combat tactics involve different methods for getting into a position to fire a missile, depending on rules of engagement, enemy threat, etc. However, there is still a significant amount of common

knowledge shared across these tactics in terms of the way they use the radar and employ missiles. This sharing falls out of decomposing complex operators into simpler ones, and using multiple rules to suggest the same operator in different situations. Such decomposition also helps avoid the problem of having rigid, scripted tactics that must be carried to completion even if they become inappropriate. Again, the reactive nature of the rules in TacAir-Soar creates behaviors that are dynamically composed by rules proposing individual operators based on the current situation. This allows the system to abandon even high-level goals if the situation warrants it.

The current version of TacAir-Soar contains over 5200 production rules, organized into about 450 operators and 130 goals. There are significant numbers of general rules that are used in many different missions, and some rules that only apply in very specific situations. Every agent loads all of the rules, but does not use them all over the course of a single mission. We chose this unified design to simplify software development – a single software system can be used for all missions, and changes and improvements to shared behaviors are automatically available across all missions.

Fly many different types of aircraft.

For STOW '97, TacAir-Soar had to control many different U.S. fixed-wing aircraft, using the appropriate weapons and sensors. The first step in addressing this problem was the construction of the appropriate parameterized models of the flight dynamics, weapons, and sensors. These models were embedded in ModSAF (Calder et al., 1993). Lockheed-Martin Information Systems and BMH Associates, Inc. built these models by defining a generic common model for each component, such as an aircraft, and then defining the parameters for a specific instantiation of a component, such as an F-14B.

The second step was to develop uniform interfaces to control these models. The Soar-ModSAF-Interface provides a uniform interface to control all aircraft using the same basic interface: all missiles are launched with the same command; and all radars deliver their information in the same format. This is not strictly true in the real world, but was a necessary simplification given our resources.

The third step was to encode general parameterized behaviors for each of these components. For example, each vehicle has parameters for cruising speeds, fuel consumption, maximum altitude, etc. Once the set of parameters and the general, parameter-driven behaviors were defined, we could add new airplanes, weapons, and sensors without changing the behaviors. Of course, occasionally some new instances of a component have new parameters that require modifications to the behaviors, but this is generally the exception.

Generate human-like behavior.

In order to be realistic and effective for training, the behavior of TacAir-Soar pilots must correspond to the behavior of human pilots (Pew and Mavor, 1998). It is not always clear how close the behavior of the synthetic pilots has to be to human behavior. Our goal is for the behavior to appear human at the tactical level. From earlier experience in cognitive modeling, we know that attempting to model human behavior to the greatest degree possible would greatly slow development and increase the required computational resources. We adopted three principles to approximate human behavior without attempting to build a detailed model (Jones & Laird, 1997).

The first principle is to develop synthetic pilots using an AI architecture (Soar) that roughly corresponds to the human cognitive architecture. Soar has a reasoning cycle that consists of parallel access to long-term memory in service of selecting and performing actions, and a goal structure that supports hierarchical reasoning. This level of behavior clearly abstracts away from the underlying implementation mechanisms (silicon in Soar, neurons in human), but captures the regularities of human behavior at the appropriate time scale for tactical aircraft – 10's of milliseconds, not nanoseconds or microseconds. Soar has been used for developing a variety of detailed models of human behavior (Newell, 1990). To speed development, we abstracted even further to modeling decisions made 2-4 times a second. This abstraction eliminates low-level eye and finger movements, but retains the basic tactical decisions. Thus, complex tactical decisions requiring multiple operators take longer than simple tactical decisions in a Soar system, just as they do in a human, providing us with a gross model of human reaction time.

Kiwi: kiwi, hawk121 your bogey is at bearing 23 for 143 angels 8
; Each plane prefaces its communication with its call sign.
; Here Kiwi is giving the bearing (23 degrees), range (143 nms) and altitude (8,000 ft).
Hawk121: Roger
Kiwi: kiwi, Contact is a bandit
; Kiwi has identified the radar contact as an enemy plane.
Hawk121: hawk121, Contact is a bandit
Hawk122: Roger
Hawk121: hawk121, Commit bearing 23 for 140 angels 8
; The rules of engagement allow Hawk121 to consider the contact hostile. At some point,
; Hawk121 decides its commit criteria are achieved and starts to intercept the bandit.
; Hawk121 uses the information from Kiwi to plot an intercept course
Kiwi: kiwi, hawk121 your bogey is at bearing 21 for 137 angels 8
; Kiwi periodically reports position information to the fighters.
Hawk121: Roger
Hawk122: Roger
Kiwi: kiwi, Bandit is closing on a hot vector
Hawk121: hawk121, Bandit is closing on a hot vector
Hawk121: hawk121, Go to defensive combat-spread formation.
; The section changes formation for the attack.
Kiwi: kiwi, hawk121 your bogey is at bearing 12 for 116 angels 8
Hawk121: Roger
Hawk122: Roger
Hawk121: hawk121, Bandit is closing on a hot vector
…
; The planes continue to close on each other until the bandit is in missile range
Hawk121: hawk121, Fox three
; Hawk121 fires a long-range missile and then performs an f-pole maneuver.
Hawk121: hawk121, Cranking right

Figure 5: Trace of interaction between automated AWACS and automated fighter pilots

While the first principle is to give the system the same basic processing structure, the second principle is to give the system the same basic sensory and motor systems. To that end, we created a "virtual cockpit" interface between our pilots and the underlying simulation. However, once again we abstracted away from the details of human vision and motor control. In TacAir-Soar we model the semantic content of the sensors available to a human pilot, and the basic commands for controlling an aircraft. There were some times when we violated this principle, which in turn compromised the fidelity of the entities' behavior. For example, in order to avoid the complex reasoning required to decide if a plane is an enemy, we allowed the radar to provide the "friend or foe" identification of an aircraft. At implementation time, it did not seem worth the development effort to encode the necessary logic for identification. Although this simplified development, it hurt fidelity, as subject matter experts noticed that our aircraft would decide to commit to intercept an enemy plane much too quickly.

The final principle is to give the pilots the same basic knowledge as human pilots. Thus, we built our synthetic pilots based on the information we extracted from subject matter experts. We did not attempt to derive new tactics, base the behavior of our agents on some general theory of warfare, or create learning systems that would attempt to develop behaviors on their own. Some of the information we encoded includes standard doctrine and tactics. However, much of it consists of the details that are not mentioned as doctrine or tactics because it is "obvious" to a human pilot. For example, we needed to encode details such as the geometric features the wingman should pay attention to when turning while in formation. Our strategy is generally to rely on SMEs. However, sometimes we have believed that the appropriate response to a situation was "obvious", and we would insert it into the system without asking an expert. Often, we would find that the situation was more complex than we thought and that what is "common sense" to an experienced pilot is quite different from the "common sense" of an AI researcher. Needless to say, a critical component of this development cycle was to have SMEs critique the behavior of the TacAir-Soar pilots.

Coordinate behavior with other entities.

In almost all missions, a plane does not fly alone but coordinates its behavior with other forces. At the immediate level, planes are organized into groups of two or four, in which they maneuver together and carry out the same mission. Small groups can combine into packages, such as a strike package, where the subgroups have different specific missions, such as escorts and attack aircraft, but the planes must coordinate their joint behavior. To simulate communications we use simulated radios to send messages (Hartzog & Salisbury, 1996; Salisbury, 1995).

Figure 5 shows an example interaction between TacAir-Soar agents. Here Kiwi is an automated AWACS controller and Hawk121 and Hawk122 are automated fighter pilots. The interactions unfold dynamically based on the situation and are not scripted. In the scenario Kiwi gets a radar contact, sends a report to the fighter pilots, and identifies the bogey as an enemy plane (a bandit). When the bandit comes within a predefined commit range, the fighters start an intercept, and eventually shoot a missile at the bandit.

Within a TacAir-Soar entity, coordination is supported by explicitly representing the different groups it is a part of, its role in each group, and the other members of the group and their roles. Knowledge about how to carry out its role in a group is encoded just like all other knowledge: as rules for selecting and applying operators. Thus, when the role of an entity in a group changes, different actions are selected and applied. For example, many of the rules in TacAir-Soar tested whether a pilot is the lead or wingman of its current formation.

In addition to coordination between synthetic aircraft, TacAir-Soar must support coordination between manned control stations, such as an AWACS or forward air controller, and synthetic aircraft. Controllers must communicate with synthetic aircraft via simulated radio. However, humans need an interface to the radio system, which we provided two approaches: a graphical user interface called the Communications Panel (Jones, 1998); and a speech-to-text and text-to-speech system called CommandTalk (Goldschen, Harper, & Anthony, 1998). In STOW '97, there were over 100 different types of messages that could be sent between TacAir-Soar agents. Examples of command and control interactions included changing stations, directing fighters to intercept enemy planes, vectoring aircraft along various desired routes, and running numerous on-call close-air support missions against a variety of targets (including ground vehicles, buildings, runways, and ships). During the exercise, humans did not have to micro-manage the planes. Sometimes when we *tried* to give the agents detailed directions, we ran into problems. We discovered that TacAir-Soar's air-to-air intercepts worked best when humans gave initial directions toward enemy planes and then shut up. The TacAir-Soar agents often had a better handle on the tactical situation than the human controllers did, sometimes engaging and shooting down planes the human controllers had missed.

Behave with low computational expense.

Another important requirement is that several synthetic pilots must be able to be run at the same time without excessive computational expense. If it cost $50,000 for each entity, then flying 100 or more entities would be prohibitive. We knew from the beginning that our approach would require more computational resources than prior finite-state automata approaches, which are more limited in the complexity of behaviors that they can generate. The question was whether that additional cost would be excessive.

To minimize costs, we emphasized efficiency throughout the design and implementation of the system. We group our approaches to efficiency into five general categories: enhancing the rule-based architecture, moving non-symbolic computation into the SMI, improving the context sensitivity of processing, adding focus of attention to sensing, and using fast but cheap hardware.

In the summer of 1992, Soar was rewritten from scratch in C (from Lisp), and significant enhancements were made to its rule matcher to improve its efficiency. The new version of Soar was approximately 15-20 times faster than the previous version. Included in these enhancements were changes that allowed Soar to match very large numbers of rules (over a million in one case) without significant degradation in performance (Doorenbos, 1993). As the project progressed, we discovered other architectural adjustments that address efficiency. One adjustment was to disable Soar's built-in learning mechanism, which improved performance around 10%. TacAir-Soar does not currently make use of learning because it provides little benefit to the expert levels of behavior that this domain requires.

The second 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. Midway through the project we realized that some of the computations being performed by rules were normally performed by a piece of

hardware on a plane instead of by the pilot. This led us to redesign our interface, moving calculations for waypoint headings and bomb attacks into a "mission computer". Such fixed, numeric, and 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 computations into the SMI.

The third technique was to restrict certain computations so that they are made only when they are going to be used in decision making. Early versions of the situation-interpretation hierarchy processed primitive input information from the SMI into as many different representations as it could. The original motivation behind this approach was to make the system easier to extend. 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, the cost was excessive. To increase efficiency, the system now uses situation-interpretation rules to compute new features only if those features are relevant to the agent's current context. For example, 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. These changes were sometimes tedious, but the purely functional concern of efficiency forced us to create a more plausible representation of human reasoning.

The fourth approach was to add multiple levels of attention to sensing. The original interface to the sensors provided the same information about every radar contact, no matter how important that contact was to the current mission. This led to many calculations in the SMI that were irrelevant. We implemented a three tiered attention mechanism to allow TacAir-Soar agents to select an appropriate level of information based on the relevance of the radar contact. For example, enemy planes being engaged receive a high attention level, while friendly planes that are not part of the mission receive low attention (once identified).

The final contribution to efficiency is the result of Moore's Law. Over the six years that TacAir-Soar has been under development, 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 a good ratio of performance to system cost, and ended up using Pentium Pros 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 six instances of TacAir-Soar on a single machine, without significant degradation in the quality of behavior. In the months since STOW '97, we have been able to increase that number.

Integration with Operational Organization

In our early work on this project, we concentrated on the design and development of TacAir-Soar and its integration with ModSAF. However, we quickly discovered that we also needed to concentrate on how people will actually use TacAir-Soar, which meant integrating it into the existing military operational organizations. In the U.S. military, all air missions are created by the Air Operation Center, which generates the Air Tasking Order (ATO) and Air Coordination Order (ACO) once a day. The ATO and ACO define all missions to be flown the next day. The ATO and ACO are generated using CTAPS, and are sent to all Wing Operations Centers (WOC) where the missions relevant to the planes at that wing are extracted. The WOC, together with the pilots flying the missions, further refine the missions. During normal operations, the ATO and ACO are generated the night before the missions are flown. Each mission is defined by over 100 parameters, and for STOW '97, each ATO contained up to 300 individual aircraft missions. Needless to say, manually typing 30,000 parameters into TacAir-Soar mission files twelve hours before takeoff was not a practical option.

Our goal is to make this process transparent to the training audience so that they use their existing tools (such as CTAPS) without modification. Our approach to mission specification is to automate the process as much as possible, but also provide the ability for human intervention at critical stages. Human intervention is necessary because we know the automation is incomplete, and we must support last minute changes to missions. In response, we created the Exercise Editor (EE; Coulter & Laird, 1996) and the Automated Wing Operations Center (AWOC). The EE allows a person to specify missions for all of the planes at an airbase using a graphical user interface. It organizes the data hierarchically according to the structure of the mission and eliminates redundant data entry for shared information. The EE has been used extensively in preparing scenarios for testing, as well as to add information that is not available in the ATO but is normally available to human pilots. As shown in Figure 6, the AWOC accepts an electronic version of the

ATO and ACO from CTAPS and transforms and reorganizes them into a form readable by the Exercise Editor. The AWOC also does some minimal reasoning to fill in missing fields.

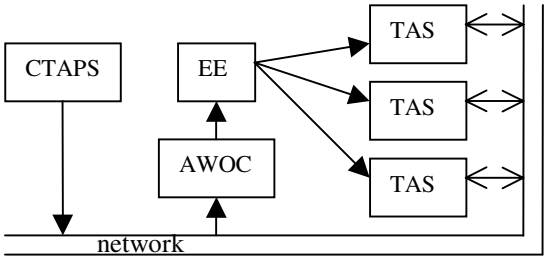

Figure 6: Diagram of mission specification software

Details of Deployment

TacAir-Soar has been used in two major exercises: STOW '97 and Roadrunner '98. TacAir-Soar is deployed at the WISSARD Laboratory on the Oceana Naval Air Station, in Virginia Beach, where it is used for continued testing of the underlying network infrastructure, as well as for evaluation and demonstration of the technology. There is also a deployed installation at the Air Force Research Laboratory in Mesa, AZ, with plans for the system's use in future training exercises and technology demonstrations.

STOW '97

In STOW '97, TacAir-Soar flew all of the missions in the ATO. For 48 hours (7am ET October 29 to 7am ET October 31, 1997), our systems at Oceana received three ATOs of 12, 24, and 12 hours each. There were 722 scheduled flights of fixed-wing aircraft, flying missions that included defensive-counter air, close-air support, suppression of enemy air defense strategic attack (air-to-ground attacks), escorts, airborne early warning, and Recon/Intel. These missions included air-to-air engagements between U.S. and opponent forces, air-to-ground attacks where TacAir-Soar entities attacked and destroyed ground targets (including vehicles, runways, and buildings), and support missions, such as intelligence gathering. During STOW '97, the training audience also requested a few missions that had not been included in our requirements, specifically attacking ships. No air-to-surface missiles had been developed for STOW '97; however, the close-air support mission was flexible enough so that we were able to dynamically retask F/A-18's to successfully attack ships using laser-guided bombs.

Each mission varied in length from 90 minutes to 8 hours, with the median being three hours. At any one time, there were from 30 to 100 planes airborne on up to 28 Pentium Pro's. TacAir-Soar flew over 15 different types of U.S. Navy, Marine, and Air Force aircraft, using all of the sensors and weapons that had been developed for those aircraft in ModSAF. The sensors included vision, radar, radar warning receiver, radio communication, IFF, and an ESM sensor. Weapons included dumb bombs, laser guided bombs, radar and infrared guided air-to-air missiles, and air-to-ground missiles. Some sensors, such as FLIR, and some weapons, such as air-to-surface missiles and guns, were not employed either because they were not supported in ModSAF, or because there was not sufficient time to build intelligent behaviors to use them.

Support personnel for STOW '97 were kept busy loading the computers with missions to fly, working on the ATO with the Exercise Editor, monitoring the planes, initiating on-call close-air support missions, and talking to visiting VIP's. However, the support personnel were never overwhelmed. Many times during the night shift, there was little to do but watch the planes fly their missions. At most times during the exercise, there were only one or two people devoted to monitoring the behavior of up to 100 TacAir-Soar entities. Although monitoring was minimized, we did discover some areas that could be further automated such as automatic distribution and loading of missions and further automatization of the processing of the ATO. Our research suggests that humans are still needed to talk to VIP's.

Problems with sensors and weapons did negatively impact the behavior of the TacAir-Soar agents. For example, the Ordnance Server, which simulated missiles in STOW '97, crashed multiple times and caused many air-to-ground missions to be ineffective because the agents lost the ability to employ maverick missiles against ground targets. A second problem arose when a component of the network failed, disabling the simulated IFF emitters, making TacAir-

Soar's planes appear to be enemies to ground forces. Therefore, ground-to-air Patriot batteries shot down a few friendly planes. Another combination of problems with ground radars and aircraft radar warning receivers prevented TacAir-Soar agents from targeting surface-to-air missile sites. This led to unrealistic losses during the early hours of the exercise, but this was not a fault of the system's knowledge or behaviors.

Overall, 99% of the missions specified in the ATO were launched. About 3% of the missions were aborted because of software or hardware failures. Many of the errors were caused by incorrect entry of commands or mission data using the Exercise Editor, the Communications Panel, or CommandTalk. More than once, a human controller vectored a plane in a direction, only to become distracted by other activities and forget about the plane, which in turn continued on the vector, sometimes outside of the battle area. This implies a need to enhance these tools to reduce the likelihood of operator error and to enhance the agent behaviors to flexibly handle incorrect or missing mission information. These errors were usually corrected either manually or through additional commands.

The performance of our software exceeded our expectations. Throughout the 48 hours of STOW '97, the behavior of the synthetic pilots was consistent with accepted doctrine and tactics. Once the planes took off, they flew all of their missions without human intervention. The planes responded to changes in weather conditions (not launching when the clouds were too low, flying above the clouds instead of through them). They landed on time and reported to the AWOC the results of their missions. In one case, an F/A-18 on a close-air support mission broke off its ground attack because it was threatened by a hostile aircraft. The F/A-18 intercepted the enemy plane, shot it down, and then returned to its close-air support mission, without any human intervention. TacAir-Soar agents did an excellent job in beyond-visual-range intercepts, where they correctly employed appropriate tactics and weapons for intercepting hostile, opponent aircraft. Throughout the exercise, human controllers were able to redirect or retask TacAir-Soar planes using the same commands (on simulated radios) as they would for human piloted vehicles.

Although TacAir-Soar's participation in STOW '97 was an "unqualified technical success," it was not a complete success in terms of training the military personnel involved. Some minor technical issues, such as difficulties with ground radars and radar warning receivers, made specific engagements unrealistic and of marginal training value. Other problems arose in getting timely intelligence reports back to the training audience. Finally, much of the training audience was unfamiliar with distributed simulation technology and although many tests were run throughout the summer of 1997, the majority of participants in STOW '97 were not involved in the testing. Most if not all of these problems can be attributed to the fact that this was the first time a distributed simulation-based training exercise the size and scope of STOW '97 had been attempted.

Roadrunner '98

From July 13-17, 1998, the Roadrunner '98 exercise was held. It was a much smaller exercise than STOW '97, focussing on integrating multiple pilot-in-the-loop training simulators together with synthetic forces to form a virtual battlespace. The goals for Roadrunner '98 were "to demonstrate the training potential of Distributed Mission Training (DMT) for improving higher order individual and collective warfighter skills, which are difficult to acquire and maintain." (Roadrunner, 1998) The exercise was distributed over seven sites, with the main sites being Air Force Research Laboratory, Mesa, AZ; Kirtland Air Force Base, New Mexico; Tinker Air Force Base, Oklahoma; and Kelly Air Force Base, Texas.

TacAir-Soar participated by flying battlefield interdiction and suppression of enemy air defense missions, in which humans in simulators flew air-to-air escorts. TacAir-Soar also engaged humans acting as opponent air forces. Overall, TacAir-Soar flew 174 missions. TacAir-Soar was not only a technical success in this exercise, but it also was operationally successful. Many of the problems that plagued STOW '97 were fixed or did not arise because of the smaller size of the exercise.

Application Payoff

To date, we have not formally evaluated the payoff of TacAir-Soar's use for military training. These evaluations will come as TacAir-Soar is involved in additional exercises such as Roadrunner '98. The dimensions for evaluation will be the cost and effectiveness of training. Although formal evaluation has not been possible, we have observed the success of the system along two primary dimensions: improved realism of automated force behavior, and economy of expense in populating a synthetic environment with automated participants.

In terms of improved realism, a number of active and retired military personnel have expressed surprise and enthusiasm at the difference between TacAir-Soar's behavior and other simulated entities. In one case, we interviewed an active

pilot who had just finished a training exercise in a flight simulator, flying air-to-air engagements against TacAir-Soar agents. The pilot was pleased to note that the system actually reacted to his maneuvers, unlike the behavior of SAFORs. He claimed that it was the best simulation training experience he had ever had.

When asked to summarize the perceived advantages of TacAir-Soar, a subject-matter expert noted the following important features of TacAir-Soar that are missing from SAFORs:

- SAFORs have no coordination with each other (or humans), so they cannot simulate large, coordinated flights, such as strike missions.
- SAFORs have no notion of different flight phases, and the different procedures that are appropriate for different portions of a mission
- There is negligible reuse of information in specifying missions for SAFORs. Each entity must be tasked individually, even for sets of similar (or identical) tasks.
- SAFORs have extremely limited facilities to react flexibly to changes in the environment or situation.

These and other similar features suggest that in many ways there is no room for quantitative comparison of TacAir-Soar with SAFORs. TacAir-Soar agents include a large number of capabilities that are qualitatively beyond anything a SAFOR can reproduce. These capabilities also lend themselves to arguments of economy. Because SAFORs have such limited capabilities, they require significant human intervention to exhibit realistic, human-like behavior. Members of BMH Associates, Inc. (our primary subject-matter experts and highly experienced users of SAFORs) estimated that one human could realistically manage controlling up to four SAFORs to produce behaviors that are acceptably flexible, reactive, and coordinated. In contrast, there were dozens of TacAir-Soar entities "in the air" at a time during STOW '97, and they only required a single person to monitor behavior and make sure nothing inappropriate was happening. Similarly, the initial specification of 722 missions for STOW '97 would have taken significantly more person-hours to complete for SAFORs than it did for the TacAir-Soar entities. These qualitative factors make it clear that TacAir-Soar provides significant savings in cost as well as more realistic simulations, although precise, quantified estimates of these benefits are still not available.

Ongoing Efforts

TacAir-Soar continues to be used for technology demonstrations and operational training at the Oceana Naval Air Station, and for training at the Air Force Research Lab in Mesa. Under the auspices of a new company (Soar Technology, Inc.) we are installing the system at additional military training sites, and continuing its development in a number of ways. One obvious area of development is to expand and deepen the 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 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 maintain consistency in their reasoning when using hierarchical architectures (Wray & Laird, 1998), the use explanation-based learning in dynamic situated domains (Wray, Laird, & Jones, 1996), how systems can acquire expert-level combat pilot knowledge through learning by observation and instruction (van Lent & Laird, 1998), how we can use TacAir-Soar to make predictions about the effects of fatigue on pilot behavior (Jones, Neville, & Laird, 1998), and how a model of emotions can improve the realism of behavior.

Acknowledgments

Paul Rosenbloom and Milind Tambe participated in the original design and implementation of early versions of TacAir-Soar, before splitting off to work on the system for rotary-wing vehicles. Karl Schwamb provided additional programming support for Soar, portions 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. TacAir-Soar was initially 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. The system continues development at Soar Technology, Inc., of Ann Arbor, MI.

References

Calder, R.; Smith, J.; Courtemanche, A.; Mar, J., and Ceranowicz, A. 1993. ModSAF Behavior Simulation and Control. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL.

Ceranowicz, A. 1998. STOW-97 - 99 In *Proceedings of the Seventh Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL.

Coulter, K. J.; and Laird, J. E. 1996. A Briefing-Based Graphical Interface for Exercise Specification. In *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation*, 113-117. Orlando, FL.

Doorenbos, R. 1993. Matching 100,000 Learned Rules. In *Proceedings of the Eleventh National Conference 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.

Goldschen, A. J., Harper, L. D., Anthony, E. R. 1998. The Role of Speech in Distributed Simulation: The STOW-97 ComandTalk System. In *Proceedings of the Seventh Conference on Computer Generated Forces and Behavioral Representation*, 621-630. Orlando, FL.

Hill, R. W.; Chen, J.; Gratch, J.; Rosenbloom, P.; and Tambe, M. 1997. Intelligent Agents for the Synthetic Battlefield: A Company of Rotary Wing Aircraft. In *Proceedings of the Ninth Conference on Innovative Applications 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 *Proceedings of the Seventh Conference on Computer Generated Forces and Behavioral Representation*, 631-635. Orlando, FL.

Jones, R. M., and Laird, J. E. 1997. Constraints on the design of a high-level model of cognition. In *Proceedings of Nineteenth Annual Conference of the Cognitive Science Society.*

Jones, R. M.; Laird, J. E.; Tambe, M.; and Rosenbloom, P. S. 1994. Generating Behavior in Response to Interacting Goals. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, 317-324. Orlando, FL.

Jones, R. M.; Neville, K.; and Laird, J. E. 1998. Modeling Pilot Fatigue with a Synthetic Behavior Model. In *Proceedings of the Seventh Conference on Computer Generated Forces and Behavioral Representation*, 349-356. Orlando, FL

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

Laird, J. E., J.; Jones, R. M.; and Nielsen, P. E. 1998. Lessons learned from TacAir-Soar in STOW-97. In *Proceedings of the Seventh Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL.

Laird, J. E.; Newell, A.; and 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.

Pew, R. W., and Mavor, A. S., Editors. 1998 *Modeling Human and Organizational Behavior: Applications to Military Simulations.* Final report of the National Research Council Panel on Modeling Human Behavior and Command Decision Making: Representations for Military Simulations, National Academy Press, Washington, D. C.

Roadrunner '98. 1998. http://www.alhra.af.mil/html/press_release.html.

Schwamb, K. B.; Koss, F. V.; and Keirsey, D. 1994. Working with ModSAF: Interfaces for Programs and Users. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL.

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

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

Wray, R. E., III; and Laird, J. E. 1998. 1998. Maintaining Consistency in Hierarchical Reasoning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Wray, R. E., III; Laird, J. E.; and Jones, R. M. 1996. Compilation of Non-Contemporaneous Constraints. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 771-778. Menlo Park, CA: AAAI Press.