AI Magazine Volume 16 Number 1 (1995) (© AAAI)

Intelligent Agents for Interactive Simulation Environments

Milind Tambe, W. Lewis Johnson, Randolph M. Jones, Frank Koss, John E. Laird, Paul S. Rosenbloom, and Karl Schwamb

■ Interactive simulation environments constitute one of today's promising emerging technologies, with applications in areas such as education, manufacturing, entertainment, and training. These environments are also rich domains for building and investigating intelligent automated agents, with requirements for the integration of a variety of agent capabilities but without the costs and demands of low-level perceptual processing or robotic control.

Our project is aimed at developing humanlike, intelligent agents that can interact with each other, as well as with humans, in such virtual environments. Our current target is intelligent automated pilots for battlefield-simulation environments. These dynamic, interactive, multiagent environments pose interesting challenges for research on specialized agent capabilities as well as on the integration of these capabilities in the development of "complete" pilot agents. We are addressing these challenges through development of a pilot agent, called TacAir-Soar, within the Soar architecture.

This article provides an overview of this domain and project by analyzing the challenges that automated pilots face in battlefield simulations, describing how TacAir-Soar is successfully able to address many of them—TacAir-Soar pilots have already successfully participated in constrained air-combat simulations against expert human pilots—and discussing the issues involved in resolving the remaining research challenges.

Interactive simulation environments are becoming ubiquitous, finding applications in education (Moravec 1990), training (Jones et al. 1993; Webber and Badler 1993), entertainment (Maes et al. 1994; Bates, Loyall, and Reilly 1992), and manufacturing (DIS Steering Committee 1994). These virtual worlds can be populated not only with humans but also with *intelligent automated agents*—AI systems that interact with humans, each other, and their virtual environment. These synthetic environments provide a new laboratory in which intelligent artifacts can be studied.

Although the concept of a synthetic environment is closely related to those of software (Etzioni 1993), robotic (Brooks 1991), and test-bed (Hanks, Pollack, and Cohen 1993) environments, it differs from each in significant ways. With respect to software environments, the most significant difference is that synthetic environments usually require real-time behavior in dynamic, limited-information worlds. A strong dependence on traditional planning techniques is thus ruled out in synthetic environments. With respect to robotic environments, the most significant difference is that synthetic environments do not have to meet the challenge of low-level perception and motor control. Thus, they enable the study of higher-level aspects of intelligence and their integration, unfettered from the demands and costs of low-level perceptual processing and robotic control. With respect to test-bed environments, the most significant difference is that synthetic environments, at least of the type focused on here, are real domains that are developed commercially for government or commercial clients who need them for their own use. Thus, developers of agents for synthetic environments do not generally have the same freedom to prestructure the environment, choose which aspects of behavior will matter, or instrument the domain for experimental purposes. One additional aspect of synthetic environments that sets them apart from these other three environment types is the com-

mon requirement that their agents act as human surrogates, necessitating an important constraint of humanlike behavior.

Our research effort is aimed at this task of producing humanlike, intelligent automated agents for large-scale, realistic simulation environments. Such agents promise several benefits that could enhance the cost effectiveness and functioning of these virtual worlds. One potential benefit is that intelligent agents could substitute for humans when large numbers of entities are required to populate a virtual world. Thus, in traffic simulators intended for training and experimentation, intelligent agents could be used to create realistic scenarios involving slow or speeding vehicles, pedestrians, traffic jams, and other complex traffic patterns (Cremer et al. 1994). Similarly, in large-scale military battlefield simulations, intelligent agents could substitute for troops who are needed to fill out an engagement but are not available in person. Another potential benefit is that artificial agents can simplify and speed up experimentation by providing more control of behavior, repeatability of scenarios, and increased rate of simulation (that is, faster than real-time simulation). This benefit can be critical for testing new products that require repeated experimentation across a wide range of situations, such as collision warning devices in traffic simulators.

Our effort began with a focus on intelligent automated agents for battlefield-simulation environments. These environments are based on *distributed interactive simulation* (DIS) *technology* (DIS Steering Committee 1994), in which large-scale interactive simulations are built from a set of independent simulators linked together by a network. Since 1983, the Advanced Research Projects Agency (ARPA) has been pursuing DIS technology aggressively as a basis for large-scale battlefield simulations. The SIMNET effort (Thorpe et al. 1989), which ARPA undertook in cooperation with the United States Army, was an important milestone in this regard. SIMNET provides a successful demonstration of the essential technologies supporting a distributed battlefield-simulation environment in which as many as 1000 independent combat entities, mainly army systems such as tanks and armored personnel carriers, can participate in training exercises in a virtual battlefield. Within SIMNET, humans and computer-generated forces—referred to as semiautomated forces (SAFORS)—can engage in unscripted, free-play simulated combat. SAFORS are able to perform many simple activities autonomously (such as driving point to point in formation or engaging an enemy) to some (relatively low) echelon of command. Higher echelons of command are vested in humans, who are also responsible for coordinating the activities of multiple SAFORS as well as intervening at lower echelons when the automated responses are inadequate (thus justifying the term *semiautomated*).

The success of SIMNET has led ARPA to pursue the extension of DIS technology across all services (air force, army, marines, and navy). The goal is to provide a cost-effective and realistic environment for training and rehearsal as well as the testing of new doctrine, tactics, and weapon-system concepts (DIS Steering Committee 1994). The extension involves expanding the types of vehicle (to include essentially all active ground, surface, and air vehicles) as well as greatly increasing the number of participating entities to a level of approximately 50,000 (thus enabling effective training at higher levels of command). Despite this large desired growth in the number of entities, cost factors dictate an upper bound of approximately 1000 on the number of humans that can directly participate. Thus, the remaining 49,000 entities required to fill out the battlefield must be computer-generated forces.

SAFORS can provide many of these automated forces. However, they have sufficient weaknesses to have led ARPA to consider alternatives. One major weakness is that the autonomy of SAFORS is extremely limited. Although they can perform simple maneuvers, SAFORS still require significant human management when it comes to the fine points of executing a complex coordinated activity, such as moving into a defensive position on a hill. This maneuver requires knowledge that SAFORS do not have about what the potential avenues of attack are, how to overlap fire for mutual protection, and how to hide behind features of the landscape. The resulting need to micromanage SAFOR entities can easily overload a human controller during the height of an engagement. Moreover, the human must attempt to simulate the combined reasoning and interactions among many independent entities in accordance with the appropriate military tactics and doctrine. Finally, the human controller must be highly skilled: It takes approximately six months to train a new controller. Such considerations led ARPA to initiate the Intelligent Forces (IFOR) Program, with the goal of developing fully automated intelligent forces that can participate in the simulated combat envi-

ronment. The ultimate idealization is to have *plug compatibility* between humans in physical simulators and computer-generated forces: Either could be plugged in to play a particular role in an engagement.

The key research challenge in building IFORs is generating intelligent, humanlike behavior in a complex, dynamic, and uncertain battlefield-simulation environment. The ideal is automated behavior that is indistinguishable from human behavior, at least with respect to the modalities through which agents can be observed (such as visual and radar sensing of the vehicle) (Tambe et al. 1994). However, pragmatically, all that is really necessary is the production of behavior that is close enough to that of humans to force the other entities in the simulation, whether human or IFOR, to interact with the IFORs in the same way they would interact with humans. This goal is still ambitious, but because this environment is synthetic, the IFORs, at least, do not need to deal with issues of low-level perception and robotic control. Furthermore, the forms of interaction between entities on a battlefield are greatly restricted by the specific goals, doctrine, and missions of the participants. For example, there are essentially no verbal interactions between opposing entities, and cooperating entities restrict their communication to the details of current missions—they rarely if ever discuss poetry.

This article explores the challenge of creating intelligent agents for distributed battlefield simulations, outlines our progress to date in meeting this challenge, and discusses the remaining research challenges. It is difficult in general to say what challenges an environment presents without understanding what tasks are to be performed in the environment. For this effort, these tasks are primarily defined by the demands of a large-scale simulated tactical exercise to take place in 1997 called Simulated Theater of War '97 (STOW-97). STOW-97 is to involve about 50,000 entities from all branches of the military. Our challenge is to provide synthetic pilots for all the missions in STOW-97 that involve military aircraft, including fighters, bombers, troop transports, reconnaissance aircraft, and helicopters.

This article begins with an examination of the simulated air-combat environment, in particular, an in-depth exploration of a fragment of a typical air-to-air mission, as might occur during STOW-97. This exploration brings out a set of requirements for automated pilots that includes a variety of important capabilities: goal-driven and knowledge-intensive behavior, reactivity, real-time performance, conformance to human reaction times and limitations, performance of multiple overlapping tasks, coordination with other agents, communication, agent modeling (plan recognition), temporal reasoning, planning, maintenance of episodic memory, and explanation.

We subsequently describe in detail the design and status of our automated pilots, along with the work that remains to enable them to participate in STOW-97. The creation of effective, intelligent, automated pilots is facilitated by an underlying architecture that can support the previously listed requirements in an integrated fashion. To develop our pilot agents, we focused on the Soar integrated architecture (Rosenbloom et al. 1991; Laird, Newell, and Rosenbloom 1987). The system we developed within Soar, TacAir-Soar (Johnson et al. 1994; Jones 1994), represents a generic automated pilot. Specific automated pilot agents are created by specializing it with specific parameters and domain knowledge. These pilot agents then participate in battlefield simulations by simulated aircraft provided by ModSAF (Calder et al. 1993), a distributed simulator that has been developed commercially for the military.

Versions of these automated pilot agents have already been demonstrated on several occasions to navy personnel, ranging from current and former navy pilots to a handful of admirals, at the United States Naval Air Station Oceana (in Virginia Beach, Virginia). Two of the earlier demonstrations focused on small-scale simulated air-combat engagements against expert human pilots. Although these engagements were simplified both in terms of the numbers of aircraft involved in the combat and the types of maneuver the human pilots could use, the automated pilots' performance was impressive enough for navy personnel to provide very enthusiastic feedback. Captain McLane, commander of the Navy Fighter Weapons School (TOPGUN), was present at one of the demonstrations and commented as follows on the set of air-combat–simulation capabilities he witnessed (of which TacAir-Soar provided the only intelligent automated pilots): "I am totally enthusiastic. I have been looking for six months for this type of capability. I want one now."

The most recent demonstration was part of a much larger engagement called Simulated Theater of War, Europe (or STOW-E), which was held 4–7 November 1994. STOW-E was an operational (and international) military

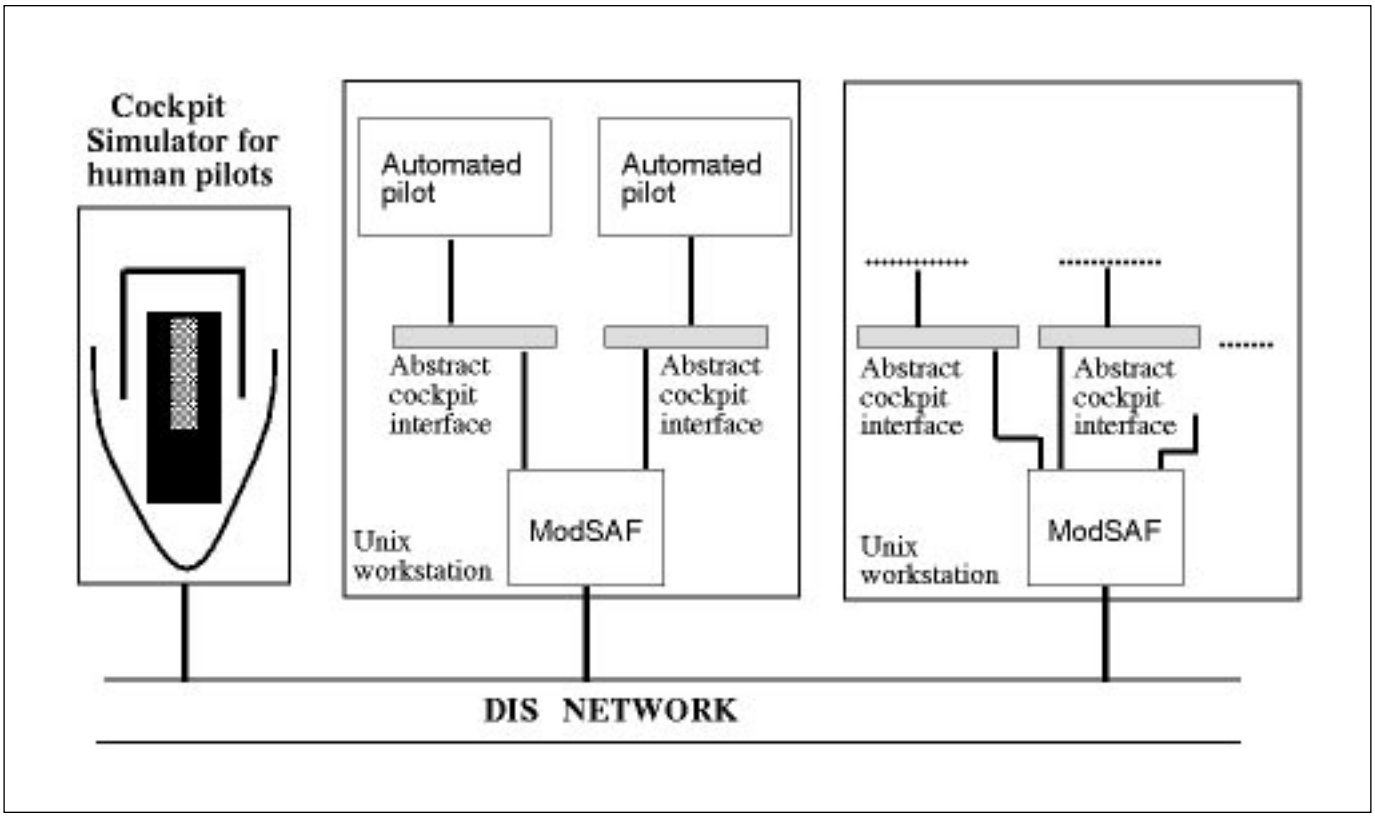

Figure 1. Human and Automated Pilots Interact with the DIS Environment Using Distributed Simulators.

exercise—in fact, the largest distributed simulation exercise to date—involving approximately 2000 entities originating from 19 sites across the United States and Europe. These entities represented everything from aircraft, ships, and ground vehicles to individual soldiers and missiles. Some entities corresponded to actual manned vehicles (for example, actual F18 fighter jets) that were broadcasting their status over the DIS network or to manned simulators. However, most of the entities corresponded to computer-generated forces. Our role in STOW-E was to provide IFORs for a range of air-to-air, air-to-ground, and air-to-surface missions. This demonstration was a real learning experience for us in understanding what happens in such complex situations, both in terms of the stress it puts on the low-level technology (such as the number of cycles required to constantly maintain and display the status of so many entities on a workstation) as well as the many surprises it provides for the agents (such as their being shot down by surface-to-air missile batteries that we had no idea existed or the unexpected forms of behavior exhibited by unconstrained human pilots). Nonetheless, we were able to successfully participate in a range of missions across all four days of the exercise, and on more than one occasion, we shot down human pilots flying in simulators. Our understanding is that TacAir-Soar is the first AI system to have participated directly in an operational military exercise.

We later provide a broader perspective on TacAir-Soar's relationship with other AI systems. The final section presents our conclusions.

The Air-Combat–Simulation Domain

In DIS, large-scale, interactive simulations are developed from a set of independent simulator nodes linked together by a network. These nodes communicate using the DIS standard network protocol (IEEE 1993). Each node simulates the activities of one or more entities in the environment and communicates relevant information to the other nodes in real time. To control the cost of the simulation, a central hypothesis underlying DIS is that there is no need to strive for completely realistic (that is, *high-fidelity*) simulations of the battlefield. Instead, the emphasis is on simulating the environment to a sufficient level (that is,

selective fidelity) to trigger tactically relevant responses from humans (Crooks et al. 1992).

Figure 1 illustrates a portion of the DIS network. It shows three interacting simulator nodes: a physical cockpit simulator for human pilots and two UNIX workstations, each running a copy of the ModSAF simulator (Calder et al. 1993). Each copy of ModSAF can simultaneously simulate a number of different fighter aircraft.[1] These aircraft can engage in simulated air combat with and against each other, either on a single workstation or across the network on multiple workstations, in real time. The total number of aircraft simulated on a single workstation depends on the complexity of the automated pilots flying the aircraft and, obviously, the type of workstation. We are currently able to run as many as six TacAir-Soar–piloted aircraft, plus ModSAF, on a single R4400-based Silicon Graphics INDIGO workstation. ModSAF models the relevant details of the aircraft, including their motion dynamics, sensors, and weapon capabilities, for selective fidelity simulation. In running the simulation, ModSAF's basic cycle starts by sequentially invoking each agent. It then updates its simulation model based on the actions of these agents, data received over the network about other agents, and the predicted movement of other agents (based on elapsed time). Finally, it broadcasts changes in its agents' motions over the network. The basic cycle time is flexible, adjusting between 2 and 15 times a second based on available computational resources.

As shown in figure 1, an automated pilot makes use of an abstract cockpit interface (Schwamb, Koss, and Keirsey 1994) to interact with the aircraft provided by ModSAF. This interface is intended to emulate the environment that a human pilot encounters—the aircraft cockpit. The interface provides an automated pilot with data corresponding to sensory information it would obtain from an aircraft cockpit, for example, radar displays, radio messages, vehicle status instruments, and visual sightings out of the cockpit canopy. However, to avoid expensive low-level signal processing, the interface provides these data in the form of numeric or symbolic encodings rather than actual images or digitized audio. For example, the interface supplies numeric data corresponding to the aircraft's altitude, say, 10,000 feet, rather than an image of its altimeter. The sensors do not provide unlimited access to the environment but instead attempt to model the perceptual limits that actual pilots would have. Currently, 31 static input and 19 dynamic input are available to an agent, which describe its own state, plus 3 static input and as many as 10 dynamic input for each radar or visual contact.

To allow the simulated aircraft to be controlled by an automated pilot, the interface also accepts commands from the pilot, again in numeric or symbolic form. These commands are then sent to ModSAF to control the aircraft's motion (desired speed, heading, altitude, rate of turn), adjust the radar system (orientation, mode), select and fire missiles, or communicate by radio. Currently, there are a total of 26 types of output command that an agent can send to the interface. The underlying model of the vehicle constrains the agent to perform only those actions that would be possible in a real plane. This model has been restricted to include some constraints on pilots. For example, the automated pilot can never cause the plane to perform maneuvers that would exceed the G force sustainable by a human pilot. One verification of the sufficiency of the cockpit abstraction is that we have been able to build software on top of it that allows humans to fly aircraft effectively in ModSAF using a simple, graphic, mouse-driven interface (van Lent and Wray 1994).

Figure 2 shows a snapshot taken of ModSAF's plan-view display of an air-combat simulation. The background shading indicates the terrain. The grid is used for measurement; in this case, each grid cell indicates an area of 10×10 kilometers. The four aircraft icons in the figure have been artificially magnified for visibility. The light-shaded aircraft are simulated F-14Ds, and the dark-shaded ones are simulated MiG-29 aircraft. The smaller windows along the side of the figure display information about the active goal(/operator) hierarchies of the TacAir-Soar–based pilots controlling these aircraft. These goal hierarchies are a small portion of the entire forest of goals that the pilots can possibly have (figure 3).

Figure 4 shows this same snapshot but with the terrain and side windows removed for clarity and an airfield added (shown by the crossed lines in the bottom left corner). This snapshot is the first in a scenario that we use to illustrate the key requirements for building intelligent automated pilots in this environment. The scenario involves four fighter aircraft: The two light-shaded aircraft are defending the airfield, and the two dark-shaded aircraft are attempting to clear out the light-shaded aircraft so that bombers can later make it to the airfield uncontested (these

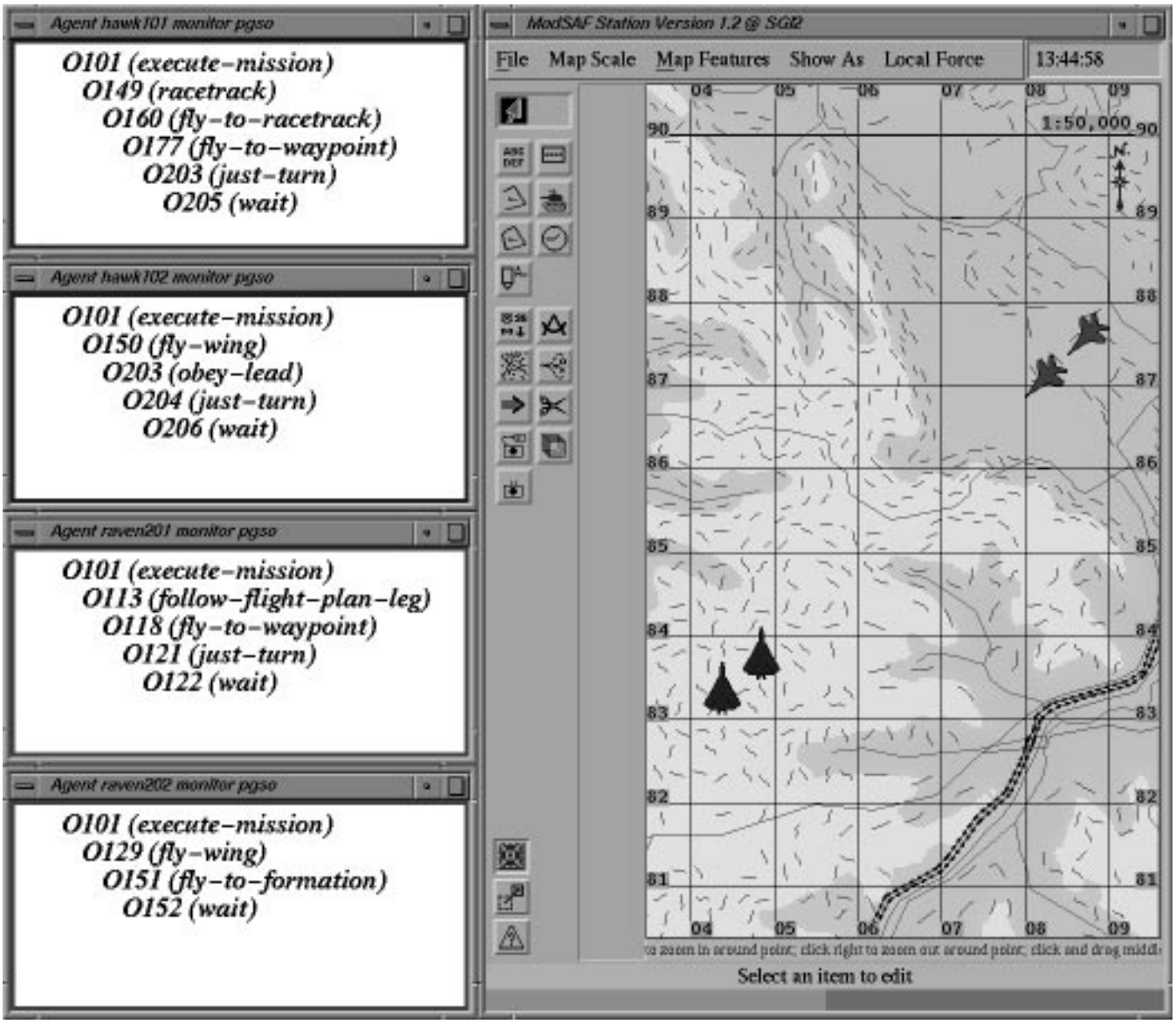

Figure 2. A Snapshot of ModSAF's Simulation of an Air-Combat Situation.

bombers are not shown in the figure). As the scenario progresses (figures 5 through 9), the two light-shaded aircraft fire missiles at the dark-shaded aircraft; however, these missiles fail to hit their targets. The aircraft then reposition themselves for a new attack. This scenario is representative of simulation runs involving TacAir-Soar–based automated pilots, although some minor modifications were made for illustrative purposes.

At the beginning, in figure 4, the two aircraft groups are separated by a large distance and cannot sense each other, even on long-range radars. To defend their airfield, the light-shaded aircraft are flying a *combat air patrol*; that is, they are patrolling in a repetitive pattern between the airfield and the anticipated direction of attack. In service of this patrol, they are flying as a closely coordinated pair, called a *section*. As specified by the mission parameters, pilot LL (that is, light lead) is the leader, or *lead*, of this section. Pilot LW (that is, light wingman) is the *wingman* of this section—it is maintaining formation and closely following any commands issued by LL. LL and LW are flying in a formation called the *fighting wing*, with LW slightly behind and offset from LL. The two dark-shaded aircraft—DW (dark wingman) and DL (dark lead)—are flying in a *trail formation*, with DW directly behind DL. DL and DW are attempting to clear out the light-

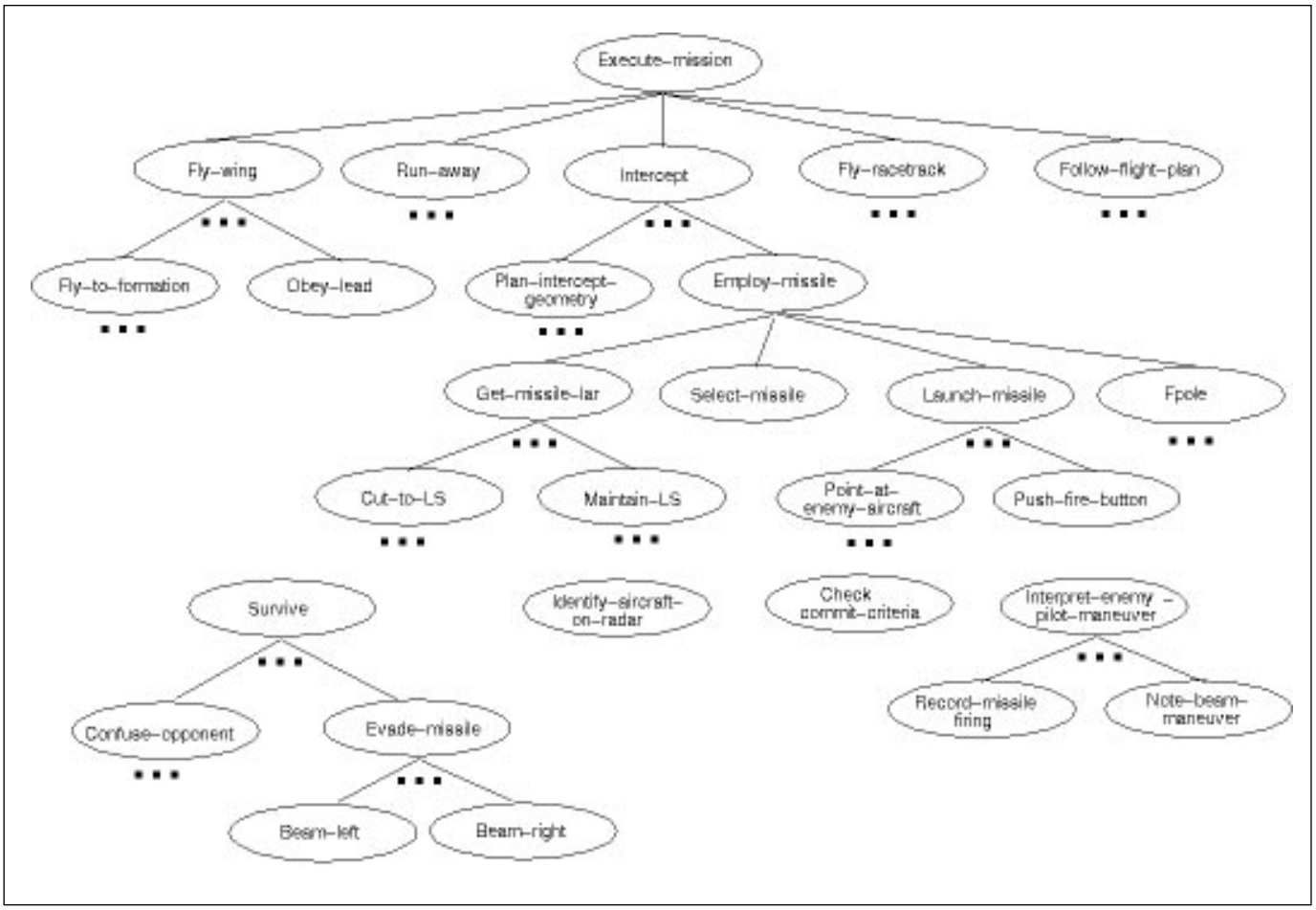

Figure 3. A Portion of an Automated Pilot's Forest of Goal Hierarchies.

shaded aircraft by performing a *sweep*; that is, they will fly in and engage LL and LW in an attempt to either destroy them or drive them away.

The specific mission and mission-related parameters for each of the pilots are specified to them in a *briefing* session prior to the mission, that is, before the aircraft are airborne. (The actual selection process for particular missions or mission-related parameters are not of concern in this scenario.) The mission parameters typically specify a pilot's role as the lead or wingman in the mission, its initial flight formation, and the specific numbers and types of missile that it is to carry on its aircraft. In addition, the mission parameters also specify the *rules of engagement* and the commit criteria, both important parameters. Rules of engagement are a set of directives regarding the general conduct of a mission. For fighter pilots, these directives might specify that they must positively identify an aircraft to be enemy military before firing. This identification can be done using a variety of methods, such as the use of special electronic beacons. *Commit criteria* are used by a pilot to decide whether to *intercept*, that is, engage in combat with, an enemy aircraft. For example, the commit criteria might specify that a pilot must intercept any enemy aircraft approaching its airfield but not intercept other enemy aircraft (because it might leave the airfield unprotected).

The important point to note in the discussion of figure 4 is that the mission and mission-related parameters help to specify a pilot's goals. Figure 3 illustrates a portion of the forest of goal hierarchies that are embodied in the pilots. Not all these goals will be active at any one time: For example, the pilot will usually only be executing one mission at a time; so, only one child of execute-mission will be active at a time. However, goals from multiple hierarchies, such as survive and execute mission, will often be simultaneously active. The pilot's behavior needs to be driven

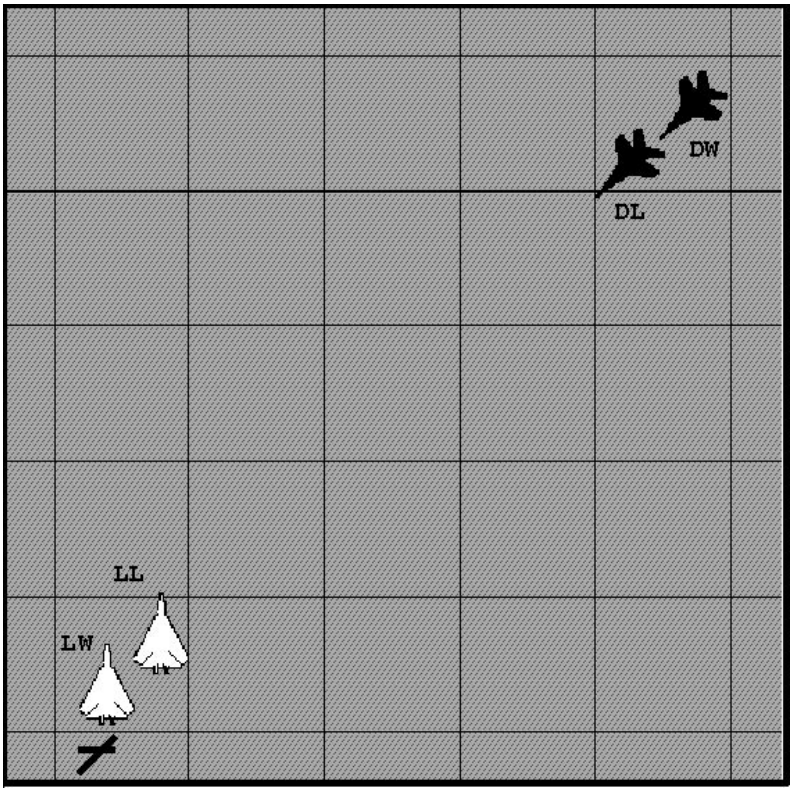

Figure 4. Snapshot 1:
The Initial Positions of the Aircraft in the Air-Combat–Simulation Scenario.

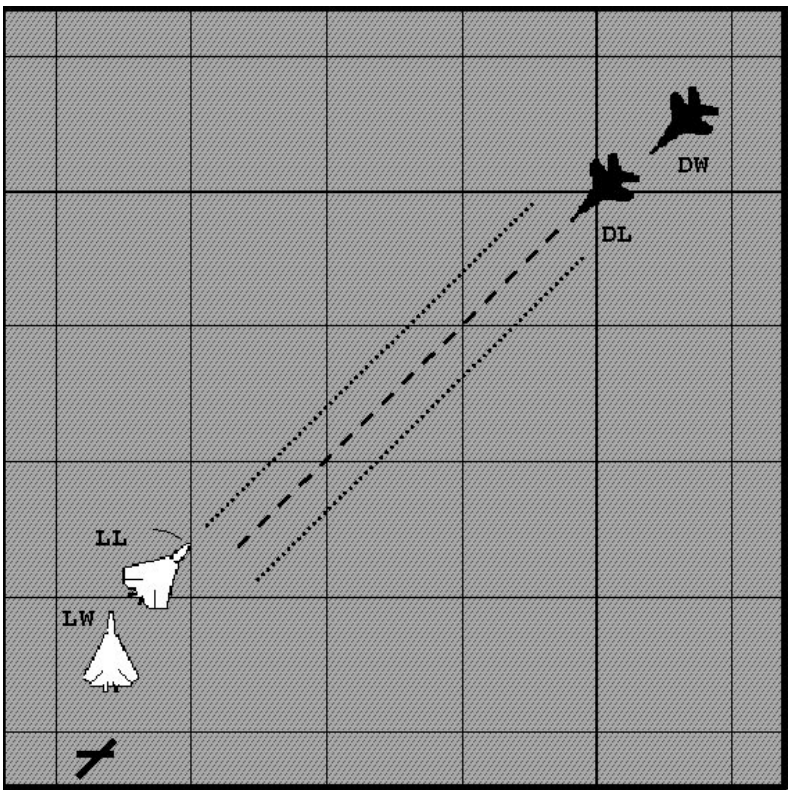

Figure 5. Snapshot 2: LL Attempts to Achieve Lateral Separation.

by these active goals. For the pilot to generate this behavior, it needs to possess a large amount of knowledge relevant to the various missions and mission-related parameters. Thus, figures 3 and 4 illustrate a requirement for knowledge-intensive and goal-driven (top-down) behavior. In addition, the previous discussion also illustrates a requirement for reactive (bottom-up) behavior. In particular, a wingman (either LW or DW) constantly needs to monitor its position with respect to its lead (either LL or DL) and continually modify its heading, speed, and altitude to maintain the formation. This level of reactivity on the part of the wingman is important; because of the realism in this simulation, continuous monitoring and correction are necessary even when attempting to fly straight toward a predetermined destination.

Now consider the situation shown in figure 5. Here, LL and LW, by virtue of their longer-range radar, spot DL and DW before they are themselves detected. As the lead of its section, LL checks the commit criteria and, according to its rules of engagement, attempts to make a positive identification of the enemy aircraft. In this case, we assume that it succeeds in doing so. In addition, because these aircraft are approaching in the direction of LL's airfield, its commit criteria are also satisfied. Hence, it decides to intercept them.

LL's first step is to ensure that while prosecuting this intercept, it maintains advantageous conditions for itself and its partner (LW). For example, it is advantageous for LL to maintain roughly 10 kilometers of lateral separation during the intercept. *Lateral separation* is the perpendicular distance between LL's position and its opponent's (in this case, DL's) projected straight line of flight. In this case, LL gains lateral separation with respect to DL rather than DW because DL is closer and, hence, considered the primary threat. DL's projected line of flight is indicated by the dashed line extending from its nose in the figure. Flying along either of the two dotted lines running parallel to this line can provide the desired 10 kilometers of lateral separation. This lateral separation is advantageous: Among other things, it makes it easier to perform a visual identification (should one be necessary), it isolates the opponents on one side of the aircraft (if done to the correct side), and it makes it easier to turn back to reengage if the opponents are passed and not destroyed. In addition to achieving lateral separation, it is advantageous to maintain high speed and altitude during the intercept (for

improved maneuverability) and not cross an enemy aircraft's projected flight path.

Based on the previous considerations, LL could conceivably generate a detailed plan, prescribing the precise sequence of actions for an optimal intercept of DL and DW. However, such a plan is almost guaranteed to fail because of the uncertainty and dynamics of this environment. For example, any turn by the opponent will invalidate the details of the plan. However, the obvious alternative of just reacting to the situation is also inadequate because the implications of the situation might extend considerably beyond the immediate choice of action. As a compromise, TacAir-Soar commits to abstract plans, such as to prosecute the intercept from DL's right side (which keeps the opponents on one side without crossing their path). Such plans constrain future actions (Bratman, Israel, and Pollack 1988) but can themselves be discarded if the situation changes enough to warrant it (such as if an attack were now required from the other side).

Once the abstract plan is established, LL attempts to position itself to fire a missile at its opponent. LL is flying an F-14D, which in this case carries two long-range radar-guided missiles as well as several medium- and short-range missiles. For LL to fire its long-range missiles, it needs to be within a particular range (distance) from the enemy aircraft. Furthermore, as just discussed, it should also have about 10 kilometers of lateral separation. As shown in the figure, LL achieves these goals by specializing its abstract plan to fly to DL's right side, along the 10-kilometer lateral separation line. (The small arc along LL's nose in the figure indicates it's turning in service of this plan.)

If at this point DL suddenly turns and increases or decreases lateral separation, LL needs to turn as well to attempt to regain the desired lateral separation. While LL is maneuvering in this fashion, LW continues to maintain its position relative to LL based on its perception of LW's motion. In general, LL will not directly communicate its maneuvers to LW but will instead expect the wingman to observe and match the turns on its own, reducing the cognitive load on LL and minimizing radio communication. The few exceptions to this rule are for particularly large turns; in which case, the wingman's understanding of the turn is more important than the communication costs.

Two important points come out of the description of figure 5. The first is, again, that the pilots' mix of goal-driven and reactive

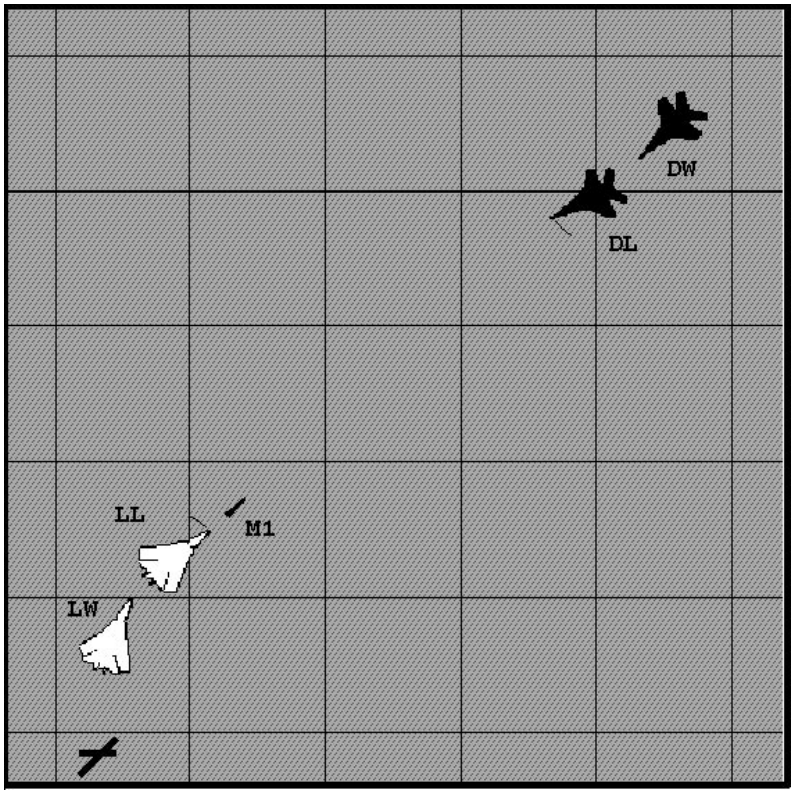

Figure 6. Snapshot 3: LL Fires a Missile at DL.

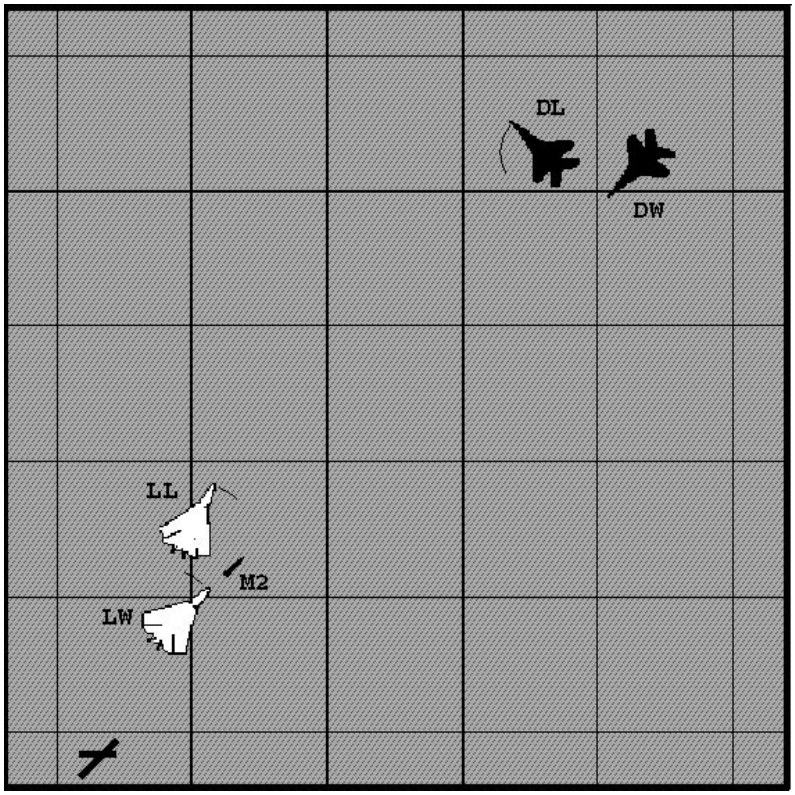

Figure 7. Snapshot 4: DL and DW Begin a Post-Hole Maneuver.

behavior, as seen in LL's decision to intercept its opponent and its continued maneuvers to maintain lateral separation. The second is that fragments of planning are also seen, particularly in LL's determination of the direction for the intercept.

As shown in figure 6, LL eventually reaches its long-range missile-firing position and fires a missile (M1) at DL. In the process, it had to decide whether it should target DL or go after DW instead. This decision is made by first selecting a *sorting criterion*, which specifies how the opponents should be divided between LW and itself. Based on the relative positions of the four aircraft, LL decides here to sort by range and communicates this criterion to LW. LW, as the wingman, then targets the furthest opponent (DW), and LL, as the lead, targets the closest (DL). LL, therefore, targets DL by getting a radar lock and turning to *attack heading*, that is, the collision course for its missile. It then fires missile M1, which obtains its guidance information from LL's radar lock. To continue to provide radar guidance to M1 and reduce the rate at which the two aircraft are closing, LL then turns its nose away from DL—a maneuver known as an *Fpole*—as shown in figure 7. As a result of the Fpole and the decreased rate of closure between the two planes, DL will not achieve its own missile-launch criteria until after M1 has reached it (the missiles of a MiG-29 do not have a range as long as those of an F-14D).

Meanwhile, back in figure 6, the aircraft come sufficiently close to allow DL and DW to acquire radar contact with LL and LW. They then maneuver themselves to reach an advantageous position in their intercept of LL and LW. While maneuvering, they observe LL's own maneuvers on their radar. These particular maneuvers are closely associated with missile firings. Therefore, DL and DW infer a possible missile firing, although they have no means of physically detecting the missile (the radar cross-section of a missile is too small to be detected), and thus, they cannot be absolutely certain about it. Nonetheless, they assume the worst and act as though a missile has actually been fired. Therefore, they suspend their intercept maneuvers and instead attempt to survive the missile. Of several available alternatives, in this case, they execute a *post-hole maneuver*. This maneuver aims not only to defeat the missile but also to confuse LL and LW. The maneuver involves DL's turning full circle so that it ends up behind DW. It also involves DL's switching roles as the lead with DW. The intended results are for DL and DW to again end up in trail formation (their previous formation) but with DW as the lead and LL and LW confused about how many aircraft are out there and where they are.

Figure 7 illustrates why the post-hole maneuver can help DL survive the missile attack. As DL begins executing this maneuver, it turns roughly perpendicular to the path of LL's radar emission. Because LL's radar operates on the Doppler principle, it cannot detect this perpendicular motion.[2] This motion causes LL to lose radar contact with DL, thus defeating missile M1 by depriving it of the required radar guidance (hence, M1 is not shown in figure 7). This loss in radar contact can also potentially confuse LL. To minimize this confusion, LL has to infer DL's ongoing maneuver based on DL's turn. LL can then anticipate the likely loss in radar contact, recognize that its missile is likely to be defeated, and switch radar modes to maintain contact (which is not to be done lightly because it guarantees that the missile will no longer track DL). During all this maneuvering, as shown in figure 7, LW reaches its own missile-firing range and fires a missile (M2) at DW.

Figures 6 and 7 again illustrate the knowledge-intensive nature of this task—the pilots need to be well informed about various maneuvers, such as the Fpole and post-hole maneuvers. More importantly, the descriptions also illustrate some new requirements. The first new requirement is *agent modeling* (Ward 1991; Anderson et al. 1990), which involves observing other agents' low-level actions—in this case, an opponent's turns—to infer its higher-level behaviors. This capability is closely related to plan recognition (Kautz and Allen 1986), although the emphasis here is not so much on recognizing other agents' plans as it is on recognizing their flexible combination of goal-driven and reactive behavior. The second new requirement is simultaneous performance of multiple tasks. This requirement is seen in DL's modeling of its opponents' actions and its simultaneous maneuvering of its own aircraft and, possibly, monitoring quantities such as lateral separation for its own intercept. A third new requirement is real-time performance. This requirement is seen in DL's need to infer a missile firing, interrupt its ongoing maneuvers, and initiate a post-hole maneuver, all before the missile gets too close. A fourth new requirement is the ability to act in a coordinated fashion. This requirement is seen in their flying and turning in formation, per-

forming coordinated maneuvers, and dynamically changing responsibilities between lead and wingman. The fifth and final new requirement is the ability to communicate with other pilots. This requirement is seen in the communication between LL and LW that occurs during the process of sorting the opponents.

In figure 8, as DL continues to turn full circle, it loses radar contact with the other aircraft. As DL turns even further (figure 9), its radar again shows three aircraft. However, DL cannot automatically assume that they are DW, LL, and LW. It is quite possible that during the time that DL lost contact, three new enemy aircraft approached the combat arena. Such a situation would be extremely unfavorable for DL. Thus, DL needs to reestablish the identities of the aircraft on its radar. In military parlance, this problem is called *maintaining situational awareness.* To this end, DL estimates the total time interval for which the aircraft have been invisible to it. Based on this estimate and the speed of each aircraft, it projects the most likely position of each aircraft. Then, based on this extrapolation, it concludes that the three aircraft on its radar are most likely to be DW, LL, and LW.

Meanwhile, in figure 8, DW, which has taken over as the lead since the beginning of the post-hole maneuver, continues to fly straight toward the opponents. Unfortunately, in this case, DW focuses its radar on LL and fails to observe LW's maneuvers. As a result, it fails to recognize LW's missile firing. Fortunately for DW, missiles sometimes simply fail to reach their targets, and this one passes by harmlessly (figure 9). Meanwhile, both LL and LW recognize that their missiles have failed to hit their targets—LL recognizes this failure because it loses radar contact with DL, while LW keeps track of the missile's expected time of impact and notices that this time has been exceeded, without its target disappearing from the radar. Furthermore, because LL and LW are neither fooled nor confused by their opponents' maneuvers, they turn back for a new attack (figure 9).

The new requirements brought out in figures 8 and 9 are centered on DL's turn. The plane that DL is flying could theoretically execute a full-circle turn extremely quickly. However, there are physical limitations on how quickly a human can turn a fighter aircraft without blacking out. Furthermore, after completing its turn, DL must identify the radar blips on its screen. Although information is available within the simulator so that this identification can be made immediately,

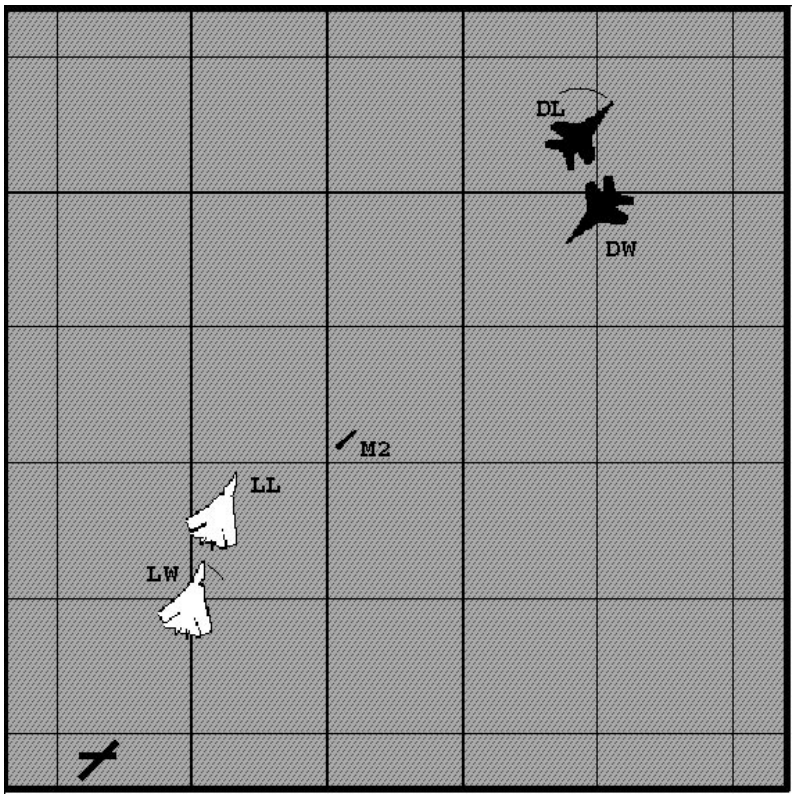

Figure 8. Snapshot 5: DL's Turn Causes It to Lose Radar Contact with All Other Aircraft.

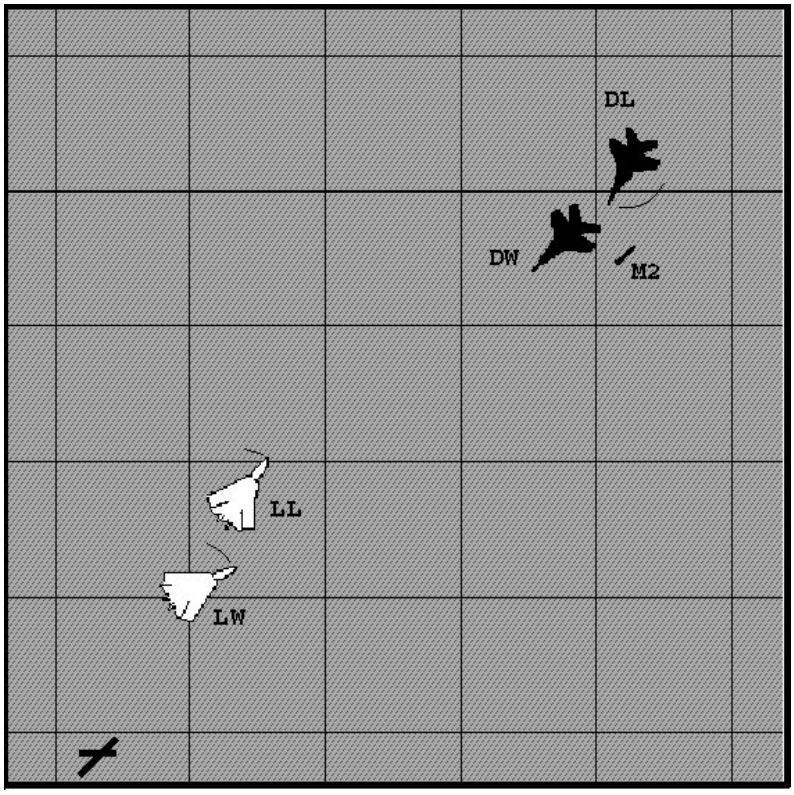

Figure 9. Snapshot 6: All Four Fighter Aircraft Have Repositioned Themselves for a New Attack

it would cause DL to react more quickly and more accurately than could humans, thus affecting the realism of the exercise. Therefore, the input available to DL are made to correspond to the same input that would be available to a real pilot, leading to DL's need to perform a mental projection to maintain situational awareness, just as humans would have to. For this projection, DL must track the time interval during which aircraft were invisible and reason with this information.

Thus, two additional requirements arise here. First, the automated pilots must conform to human limitations on aspects such as G forces, reaction times, and levels of accuracy. Second, the automated pilots need to reason with temporal intervals. This reasoning with temporal intervals also shows up in LW's tracking of its missile's expected time of flight as well as in other situations in this environment.

Typically, the scenario described here would continue beyond the snapshot depicted in figure 9, and the pilots would continue to employ different maneuvers until either they themselves or their opponents were destroyed or ran away. Following the scenario, they would participate in a debriefing process in which they would be asked by an instructor, an evaluator, or a superior officer about what happened during the mission and why they did what they did at key junctures (Johnson 1994a, 1994b). Debriefing is an important means by which others can validate and gain confidence in the pilot's judgment. With respect to capabilities, debriefing reinforces the need for communication, although this time in a more expansive and less time-critical setting. It also reveals the need for some form of episodic memory to enable the pilots to remember what transpired during their engagements and, most critically, a self-explanation capability. Such a capability is essential for automated pilots because it can help give military personnel confidence in the agents' knowledge and reasoning capabilities. Although debriefing can also potentially provide feedback from which the pilot can learn, such learning doesn't actually come up as a requirement in this scenario because these are expert pilots performing a routine mission. However, such learning is likely to become increasingly important in the future.

The scenario that we just went through is only one of many possible scenarios, and a large number of variations are possible. Nonetheless, the snapshots and associated discussion bring out most of the key requirements for building automated pilots in this virtual environment, at least within the space of simulation parameters that we have encountered so far. In summary, these requirements are (1) goal-driven behavior, (2) knowledge-intensive behavior, (3) reactivity, (4) real-time performance, (5) conformance to human reaction times and limitations, (6) overlap of performance of multiple high-level tasks, (7) multiagent coordination, (8) communication, (9) agent modeling (especially opponent modeling), (10) temporal reasoning (dealing with time intervals), (11) planning, (12) maintenance of episodic memory, and (13) explanation.

Additional requirements are also expected to come up as these requirements are addressed and as new types of vehicle (such as helicopters) and mission are investigated. For example, learning is eventually going to be required to improve performance both during and across engagements. However, because learning doesn't come up here to any great extent, we won't focus on it further in this article other than to mention that using it effectively in this domain looks to be quite nontrivial.

Creating Intelligent Automated Pilots

To create effective automated pilots, it is necessary to address the list of requirements from the previous section in an integrated fashion. We have chosen to do so using Soar, a unified software architecture being developed as a basis for integrated intelligent systems (Rosenbloom et al. 1991; Laird, Newell, and Rosenbloom 1987). Soar is also a developing unified theory of cognition (Rosenbloom and Laird 1994; Newell 1992a, 1992b, 1990). This theory entails a number of constraints (for example, concerning human reaction times, knowledge representation, and learning) that bear directly on potential solutions to the requirements presented previously. For readers unfamiliar with Soar, a brief description appears in the sidebar.

TacAir-Soar has been constructed from Soar through the addition of perceptual and motor interfaces (in the form of C code) that allow TacAir-Soar pilots to fly ModSAF planes in the DIS environment, specific knowledge about the tactical air-combat domain (in the form of rules), and strategies for aiding in addressing those requirements not fully addressed by the Soar architecture itself (also in the form of rules). This interface and knowledge are currently maintained as a con-

stant across all TacAir-Soar–based pilots. Individual pilots can then vary in terms of the mission and vehicle parameters provided to them during the briefing process and, to a small extent, the knowledge and tactics that they employ (by selectively disabling parts of less competent pilots). Allowing wider variations in structure, knowledge (including tactics), temperament, and preferences is a topic for future work.

The first two required capabilities—goal-oriented behavior and knowledge-intensive behavior—are addressed by the manner in which TacAir-Soar uses its architecture and knowledge in the process of dynamically expanding and contracting a goal hierarchy for this domain. Let's look at a concrete example of LL's operation from a situation that arises shortly after it has decided to prosecute the intercept of DL and DW (figure 5). Figure 10 depicts LL's hierarchy of problem spaces and operators. (Goals are represented implicitly in this diagram as the desires to apply operators that have reached an impasse.) This hierarchy is generated as follows:

In the top-most problem space (top-PS), LL is attempting to accomplish its mission by applying the *execute-mission* operator. The termination condition of this operator is the completion of LL's mission (which is to protect its home base for a mission-specified time period). Because this condition has not yet been achieved, a subgoal is generated to complete the application of *execute-mission*. The execute-mission problem space is selected for use in this subgoal. In this problem space, LL selects the *intercept* operator to perform the intercept. The termination condition of this operator—that the opponents are either destroyed or chased away—is also not yet achieved, leading to a second subgoal of completing the intercept. The intercept problem space is selected for use in this goal. Within this problem space, LL selects the *employ-missile* operator, which uses missiles to either destroy the opponents or force them to run away. However, because LL has not reached a position from which it can fire a missile, a third subgoal is generated. The employ-missile problem space is selected for use in this subgoal. Within this problem space, LL attempts to achieve a firing position for its missile by selecting the *get-missile-LAR* operator. (LAR stands for *launch acceptability region*, that is, the region from which LL can effectively fire a missile at its opponents). Because such a position has not been achieved, a fourth subgoal is generated, and the get-missile-LAR problem space is selected for it. Here, the *cut-to-LS* operator is selected to achieve lateral separation. Finally, the *cut-to-LS* operator leads to a fifth subgoal, within which the desired-maneuver problem space is selected. Within this space, the *just-turn* operator is selected and applies directly to issue a new heading command to the cockpit

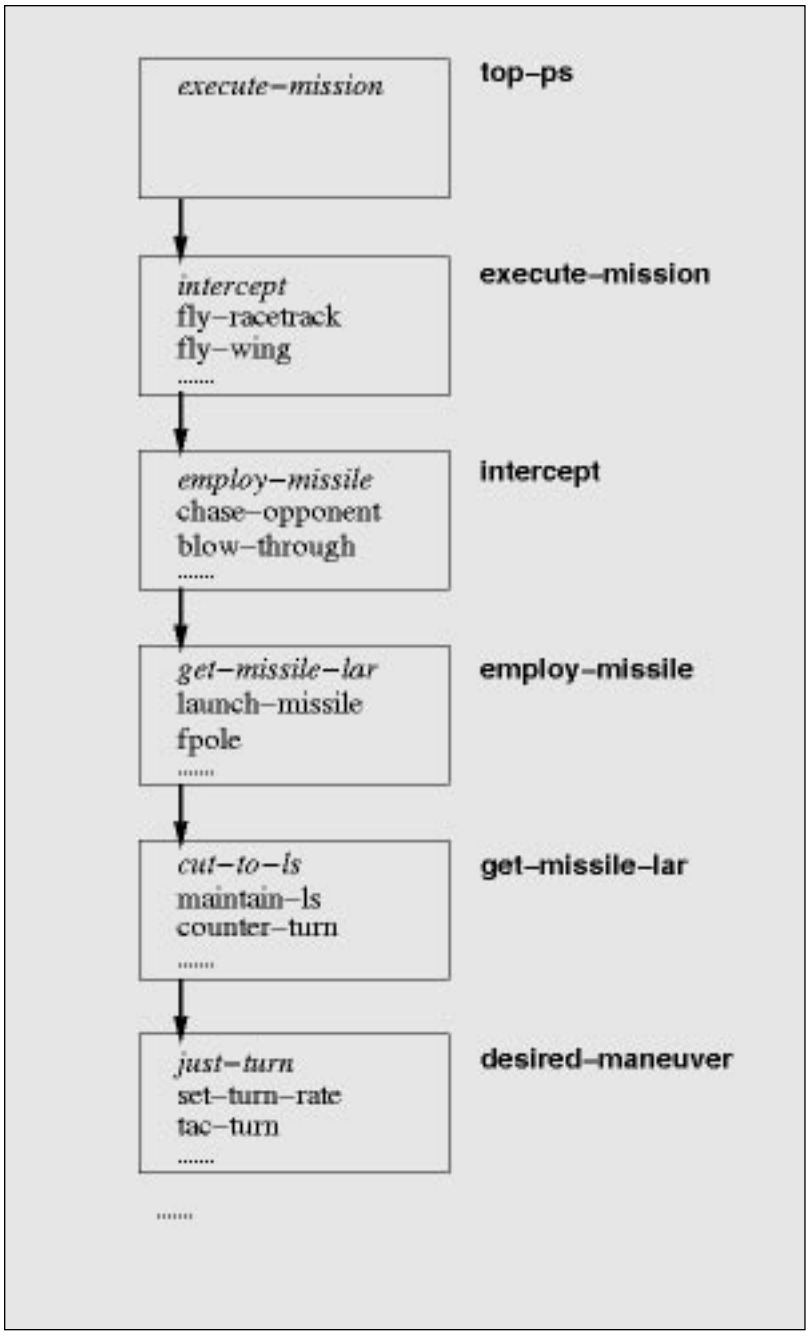

Figure 10. LL's Problem-Space Hierarchy.

Boxes indicate problem spaces, with names in bold to their right. Names within boxes indicate operators within problem spaces. Italicized names indicate currently selected operators.

interface, causing the aircraft to turn in an attempt to achieve lateral separation.

As LL turns, if the termination conditions of any of the operators in the hierarchy are satisfied, then the operator is terminated, and all the subgoals generated in response to this operator are removed—they are now irrelevant. For example, if the desired lateral separation is achieved, then the termination condition of the *cut-to-LS* operator is achieved. The operator is then terminated and replaced by the *maintain-LS* operator—to maintain lateral separation—and its subgoal and associated problem space (desired-maneuver) are flushed. Similarly, if a missile-firing position is reached, the termination condition of LL's *get-missile-LAR* operator is satisfied; hence, it is terminated along with the two subgoals beneath it.

The goal-driven aspect of this behavior should be obvious from the example. The knowledge-intensive aspect arises because all TacAir-Soar's behavior is driven by accessing relevant knowledge about action. The generation of alternative operators occurs through rules that examine the goal, the problem space, and the state and create instantiated operators that might be relevant. Once these operators are generated, further rules fire to generate preferences that help select the operator most appropriate for the current situation. Once an operator is selected, additional rules fire to perform the appropriate actions.

The requirement for reactivity is addressed through a combination of Soar's use of productions to enforce ubiquitous context sensitivity in the representation and use of knowledge as well as Soar's open and integrative decision procedure. TacAir-Soar's productions are constantly testing input from sensors (such as the heading of an opponent), inferences drawn by other knowledge about collections of sensor data (such as an opponent is a threat), current goals and operators being pursued (such as intercepting the opponent), and potential actions to take in this situation (such as achieving a position to fire a missile). This knowledge can react to changes in the situation (such as a sudden turn by its opponent) by suggesting new operators to pursue at any level in the goal hierarchy, generating preferences among suggested operators, terminating existing operators, or even directly suggesting new external actions. When the productions have finished making their suggestions (generating their preferences), the decision procedure integrates all available suggestions/preferences together to determine what changes should actually be made to the goal/operator hierarchy. Thus, at every step, TacAir-Soar's behavior is based on its assessment, given the current situation and its own goals, of what is the best thing to do.

Although using rules to provide reactivity is similar to the approach taken in pure situated-action systems (Agre and Chapman 1987), this domain demands—and TacAir-Soar enables—the use of persistent internal state. Relying only on externally available cues is simply insufficient here for effective performance. Simple examples of this phenomenon can be seen in situations where other aircraft are not visible on an agent's radar. For example, LL loses contact with DL in figure 8 but needs to reorient itself to fire a second missile at DL. If LL waits for DL to reappear on its radar, it might have already lost its advantage. Similarly, DL loses contact with the other aircraft in figure 7 but needs to turn a full circle. As discussed earlier, DL cannot just keep turning until it sees some aircraft back on its radar because other aircraft might be in the vicinity. In such cases, TacAir-Soar can engage in a substantial effort to deliberately maintain internal state information.

The basic real-time performance requirement is that simulated pilots need to be able to respond to environmental contingencies as fast as human pilots do; otherwise, the behavior can be unrealistic. Although combat between jet aircraft sounds like the epitome of a high-speed, short-deadline task, it turns out that it mostly is not. In particular, at least in the initial phases of the combat, given the great distances involved, pilots do have seconds or tens of seconds available for their initial activities, such as identifying contacts on radar or planning intercept geometry. However, even at these distances, there can be short bursts of intense activity, such as maneuvers to evade missiles, and unrealistically slow reaction times can be fatal. As the aircraft continue to maneuver and get closer to each other, slow reaction times become increasingly problematic. Such problems can arise either from too many decisions being required before action occurs or too much time being required for each decision. We have approached the former problem by focusing on expert-level performance in TacAir-Soar. In particular, we have tried to provide it enough knowledge to avoid the need for extensive metalevel reasoning and planning, especially during time-critical segments of an engagement. This expert-level knowledge allows operators and actions to be selected without impasses occurring and, therefore, without extensive numbers of deci-

sions. The latter problem—individual decisions taking too long—can potentially occur because of slowness in the ModSAF simulator, the Soar architecture, the interface code between ModSAF and Soar (that is, the cockpit abstraction), or the use of TacAir-Soar's knowledge. There is little action to be readily had from the first three sources: We have no direct control over the ModSAF simulator, the Soar architecture is already heavily optimized with a new C-based version that is a factor of 15 to 20 times faster than the previous Lisp-based version (Laird et al. 1993), and the interface is quite minimal. We have thus focused our efforts on the fourth source—the knowledge (that is, the rules). This effort has involved using rule-coding styles that help ensure inexpensive rule match; detecting and eliminating unnecessary rule firings; and moving some low-level, bottom-up computations out of the rules and into the interface. To illustrate moving computations from rules into the interface, consider what information the plane's radar provides to a TacAir-Soar–based pilot. Initially, this information consisted of only the position, heading, and speed of detected planes. Productions would then compute relevant relations, such as the bearing to the plane, its lateral separation, and the bearing of the agent's plane from the other plane (these relations are relevant for certain tactics).

However, these values needed to be recomputed each time the planes moved (essentially every decision); so, the calculations were moved into the interface where they could be done more efficiently with dedicated C code, and the relations thus become part of the input from the sensor. This approach might seem inappropriate. However, it turns out that designers of fighter aircraft have also heavily used this strategy. They attempt to reduce the pilot's cognitive load as much as possible by offloading many of the low-level, bottom-up computations onto the plane's instruments.

In addition to the basic real-time requirement, there is also a secondary real-time requirement here that stems from how the ModSAF simulator interacts with the synthetic pilots. ModSAF's control structure is based on the assumption that each agent can be trusted to relinquish control of the processor on its own and, therefore, that each agent can be free to take complete control of the processor while it is active. Therefore, during each cycle of the simulation, each agent is called in turn to perform actions until it relinquishes control. Once all the agents are done, the environment is updated. ModSAF depends on each such cycle—executing all the agents plus updating the environment—completing in 66 to 500 milliseconds total (the underlying simulation dynamically adjusts to the time required), although it can be longer for short periods of time without disrupting the simulation. To cope with these implicit soft deadlines and, thus, deserve the trust that ModSAF is implicitly placing in the TacAir-Soar–based pilots, we have limited each pilot to one decision in each simulation cycle and utilized the techniques described earlier to minimize the amount of time required for each decision. The softness of the deadlines in this domain makes this approach acceptable, as long as the system itself is fast enough, and the worst-case behavior is not too far out of line.

The bottom line with respect to the real-time requirement is that on a single Silicon Graphics R4400-based INDIGO workstation, we are currently able to run the ModSAF simulator plus as many as four TacAir-Soar–based pilots and remain within plausible cognitive limits 94 percent of the time and within the simulator's expected time limits 100 percent of the time and as many as six pilots and remain within plausible cognitive limits 75 percent of the time and within the simulator's expected time limits 88 percent of the time. To go beyond six aircraft scenarios, additional aircraft can be added by distributing them across multiple workstations. As the need for synthetic pilots expands to hundreds (or thousands), the question of how much hardware is required for each pilot will become critical, suggesting that additional speedups will always be useful.

The requirement of conforming to human reaction times is closely related to the real-time requirement. In fact, the basic real-time requirement is just that synthetic behavior be not too much slower than the corresponding human behavior. However, there is also a symmetric requirement that synthetic behavior be not too much quicker than the corresponding human behavior. To some extent, the Soar architecture starts us off in the right place in terms of how long it takes to do simple cognitive tasks (Newell 1990). In practice, there has not been a significant problem with cognition running too quickly. What has proven to be a greater problem is the rate at which human physical activities are simulated. Because we are not using a model of the human body and, in particular, of the speed at which it can perform physical activities such as moving eyes, scanning dials and

radar screens, and turning knobs, TacAir-Soar pilots can react significantly more quickly than human pilots. Pragmatically, such problems can often be ignored because they are effectively *masked*—that is, reduced to the level of noise—by the much longer response times of the plane or environment. For example, the time to respond to updated radar information might be dominated by the refresh time of the radar itself, which can require from 2 to 10 seconds. In other cases, we have introduced deliberate delays to slow down some of TacAir-Soar's responses. Similar ad hoc approaches have also been taken to model other human physical limitations; for example, maximum G forces sustainable by human pilots are modeled by restricting the plane so that it cannot exceed them (even if the plane theoretically could). The correct long-term solution to these problems, of course, would be to use a better model of the physical body.

The remaining requirements from the list presented earlier—overlapping performance of multiple high-level tasks, agent modeling and coordination, communication, temporal reasoning, planning, maintenance of episodic memory and explanation—are ones for which the underlying Soar architecture does not either directly provide a solution or dictate a specific approach to a solution. These requirements have thus presented interesting research challenges that have to be addressed, at least to some extent, to provide competent behavior in this environment. In many ways, this research has been made easier by the generalized support Soar provides for intelligent behavior, as discussed under the capabilities already described. However, Soar also imposes a number of constraints on development—stemming from its being a theoretically motivated attempt at constructing a coherent architecture for intelligent behavior rather than simply a grab bag of tools—that don't always fit precisely with the demands of this environment. These mismatches either reflect ways in which the Soar architecture is inadequate, or they reflect constraints within which people do—and synthetic systems should—find ways of existing. Which of these is the case is an open question, although an actively investigated one.

The first of the remaining requirements—simultaneous performance of multiple high-level tasks—has turned out to be the most problematic of the set. In particular, as limited by the Soar architecture, TacAir-Soar cannot directly construct multiple independent goal hierarchies in service of multiple high-level tasks. For example, consider a situation where LL needs to interpret an opponent's maneuver while it attempts to reach its own missile-firing position. In this case, LL constructs a goal hierarchy for achieving missile-firing position, as shown in figure 10. The problem is that although a second goal hierarchy is also required here—to interpret the opponent's maneuver—LL cannot independently construct it because of the constraint imposed by Soar that there be only a single active goal hierarchy. The approach to this problem that is currently implemented in TacAir-Soar is to commit to just one of the goal hierarchies and install operators for the remaining high-level goals, as needed, at the bottom of this goal hierarchy. Thus, in the earlier example, LL commits to a hierarchy for attaining a missile-firing position but then installs an interpret-maneuver operator as a subgoal of the just-turn operator. Although with sufficient care this approach can be made to work, it also has a significant problem: Any attempt to update the upper goal hierarchy will eliminate the lower one because Soar automatically interprets *lower* to mean *dependent on*, even though the hierarchies really are independent here. We generally attempt to minimize this problem by ensuring that the lower hierarchy is present for only a brief time—in particular, it is usually limited to the application of a single operator within a single decision—however, a more general solution is clearly required. Alternative solutions that are currently under investigation include (1) changing the architecture to allow the explicit use of a forest of goal hierarchies (Jones et al. 1994); (2) merging operators across goal hierarchies so that a single hierarchy can represent the state of all the hierarchies simultaneously (Covrigaru 1992); and (3) flushing the current goal hierarchy whenever a new one needs to be established yet depending on learning to compile goal hierarchies into single operators and, thus, reduce (or eliminate) the overhead of constantly flushing and rebuilding large goal hierarchies (Rubinoff and Lehman 1994).

The next requirement—agent coordination—is an extremely difficult problem in general (Durfee and Montgomery 1990); however, the military simplifies it as much as possible by relying on shared knowledge of common doctrine and tactics and providing detailed mission specifications that outline ahead of time the roles of the coordinating pilots. Most of this knowledge is already encoded in TacAir-Soar agents so that they can correctly perform the missions, although

some additional knowledge is required about the capabilities of other agents. However, there are still many circumstances in which dynamic coordination is necessary. For example, a more realistic version of the scenario described earlier would include an E2C aircraft (which has a radar that can see in a 250-mile radius) aiding LL and LW and a ground controller (also with a large radar) aiding DL and DW. These additional agents support the fighters by providing them with sightings of planes and sometimes directives for their missions. In other situations, a controller might dynamically change a plane's mission. Participation in these types of scenario requires the TacAir-Soar agents to communicate and simultaneously maneuver their aircraft (a capability that is currently achieved using the technique outlined in the previous paragraph). It also requires the agents to be *taskable*, that is, to be able to dynamically change their missions (and goals) in response to interactions with other agents. Current TacAir-Soar agents do participate in these types of scenario both as pilots and controllers (Laird, Jones, and Nielsen 1994).

The primary communication requirement in the scenario arises in the context of interagent coordination. This form of communication is currently based on a fixed set of message templates that are derived from the actual messages spoken by pilots (there are over 70 different types of message that TacAir-Soar agents can send and receive). Although this approach has proven adequate in the scenarios encountered so far, its lack of flexibility will likely prove problematic in the future. To provide for flexible natural language communication, a research effort is under way to integrate NL-Soar (Lehman, Lewis, and Newell 1991), an independently developed, Soar-based, real-time natural language system, into TacAir-Soar (Rubinoff and Lehman 1994).

A secondary communication requirement in the scenario arises during postmission debriefing (Johnson 1994a, 1994b). Here, TacAir-Soar must accept questions and generate responses. Questions are currently input through a dynamically generated set of menus. Answers are generated using a combination of text (with a simple functional-unification-grammar–based language generator implemented in Soar) and graphic depictions of aircraft configurations. The communication techniques used here, although more general than those currently used for interagent coordination, still need to be generalized. These two forms of communication capability also need to be merged into a single multifunctional capability.

The next three requirements from the list—agent modeling, planning, and temporal reasoning—are also at the top of our research agenda. For each of these capabilities, the real-time, dynamic, limited-information, multiagent nature of this environment poses the important research challenge. For example, much of the existing work in agent modeling, as well as in the closely related area of plan recognition, has focused on static, single-agent environments (except perhaps for a recent effort [Rao and Murray 1994] also in the arena of air-combat simulation). The techniques that have emerged from this work are not directly applicable in our environment given the agents' complex mix of goal-driven and reactive behavior. The key idea in the solution that we are investigating is that the mechanisms that a TacAir-Soar agent employs in generating its own flexible and reactive behaviors can be used to track other agents' flexible and reactive behaviors (Tambe and Rosenbloom 1994). Preliminary solutions from these investigations are being incorporated into TacAir-Soar.

The next requirement in the list is maintaining episodic memory. Episodic memory in TacAir-Soar is used primarily to support explanation—an agent cannot explain its decisions and actions if it cannot remember them. It is also employed to a limited extent during agent modeling, so that an opponent's current actions are interpreted based on its actions in the past. Episodic memory in general raises hard learning issues for Soar: Because episodic memory is a basic characteristic of human cognition, a unified theory of cognition ought to provide a general mechanism for it. The episodic-memory model in TacAir-Soar was designed to meet specific requirements for agent debriefing as well as general constraints imposed by the domain and the architecture. The general constraints are that episodic memory should add minimal processing overhead during mission execution and, for efficiency reasons, should not substantially increase the size of Soar's working memory. Minimizing processing overhead means that episodic memory construction should not involve the explicit application of operators that can compete and interfere with the other tasks that the agent is performing. The need to minimize the use of working memory implies that simple solutions, such as logging all events and state changes, are inadequate. To provide a high-quality debriefing capability, it is important that the agent have an accurate memory of

events and their sequence, perhaps more accurate than human episodic memory, which is prone to errors in both reconstructing events and determining their proper order. The current implementation of episodic memory is a compromise among these requirements. The sequencing of events (that is, significant operator applications) is recorded in working memory so that it can be recalled accurately. The states in which these events occurred, however, are stored by committing state changes to long-term memory. The long-term memory mechanism employs chunking, using a variant of the data-chunking mechanism used in other Soar systems to remember declarative information (Rosenbloom, Laird, and Newell 1987).

The final requirement—the ability to explain one's own behavior—is both constrained and facilitated by the Soar architecture. The constraint arises from the inability of Soar-based agents to directly examine their own rules. This constraint is motivated by, among other things, comparable human limitations on self-reflection. It implies that understanding of one's own behavior must come from the observation of, the learning about, and, possibly, the experimentation with this behavior rather than by direct examination of the code that generated it. The facilitation arises from the metalevel support Soar provides for exploring its own behavior. Such support enables Soar to both reflect on the actual decisions that were made and reason hypothetically about alternative situations and decisions to identify which situational factors are most significant for decision making. In contrast to the time-limited nature of the behavior required during missions, where extended metalevel activity can be problematic, such behavior is just what is needed during a debriefing session. Furthermore, the results of meta-level reasoning are summarized in chunks, which facilitate subsequent explanation. Currently, TacAir-Soar's explanation capability allows it to answer questions regarding the actions it performed and the conclusions it came to during task performance. In the future, additional capabilities might enable TacAir-Soar to obtain feedback from expert pilots during such an explanation session, providing TacAir-Soar an opportunity to improve its performance by learning from this feedback.

Current Status and Future Needs

TacAir-Soar is at present capable of participating in simulated tactical air-combat scenarios involving fighters that are either on their own or are flying in coordinated pairs (that is, as a section). They might additionally be supported by ground or air controllers who can use their much larger radar coverage to provide information to the fighters on enemy fighters that are outside their own radar range. These controllers can be realized either by humans through a special graphic interface to ModSAF or by the specification of a ground- or air-control mission to a TacAir-Soar agent.

By taking advantage of the networked environment (figure 1), we have been able to run scenarios involving as many as eight fighters and two controllers. TacAir-Soar's main area of expertise is *beyond-visual-range combat*, where pilots only have radar and communication information about enemy aircraft, as opposed to *within-visual-range combat*, where pilots can also directly see the enemy aircraft. TacAir-Soar pilots can fly three different types of aircraft, three different types of "aircraft-versus-aircraft" combat mission, and a substantial variety of maneuvers and tactics. TacAir-Soar includes approximately 200 operators spread over 24 problem spaces and about 1700 rules. It has consumed about two years' worth of team effort to reach this point (including a large ramp-up time for relevant knowledge acquisition and infrastructure development).

More recently, we have been extending TacAir-Soar to fly types of aircraft and perform types of mission that are quite different from those used in "aircraft-versus-aircraft" scenarios. With respect to aircraft, the key new type is an *attack helicopter* (Gunston 1986), which acts much like a hybrid between a fixed-wing aircraft and a ground vehicle (such as a tank). Its missions, tactics, and weapons differ in a number of significant ways from fixed-wing air-to-air combat, most importantly in the way it needs to reason about, plan with respect to, navigate within, and exploit terrain features such as hills and trees. With respect to fixed-wing aircraft missions, the key new one is *close air support*, which is an "aircraft-versus-ground target" mission conducted in support of ground troops. A key additional capability that TacAir-Soar pilots require for close air support is the ability to accept new missions dynamically and plan the details of its attack. This capability involves determining factors such as the timing, ordnance, altitude, and method of attack. This planning is based on information regarding the weather, terrain, target size, enemy antiaircraft capabilities, and so on. Thus, the behavior of the system

can be very different if one of these parameters is changed (for example, if the weather is partly cloudy instead of clear). This planning capability is a prototype for some of the types of planning that will be required for other missions.

During the next three years, we will expand the scope (breadth) and competence (depth) of TacAir-Soar in preparation for the STOW-97 exercise. In terms of scope, TacAir-Soar needs to cover a total of 16 different mission types, including suppression of electronic defenses, strategic attack, interdiction, surface attack, troop transport, reconnaissance, and refueling. Some of these mission types are natural extensions of its air-combat and close air support missions, but others are quite different.

Providing competence in these missions is not just a matter of developing each mission for an individual plane to fly. Many of these missions, such as close air support, require a section or division (two sections) of planes to check in with various controllers who can dynamically modify various aspects of the mission, such as the general route the planes are to take or the specific target of the attack. Moreover, there are integrated missions, such as an offensive strike, that involve as many as 20 to 30 aircraft, each with its own mission but able to coordinate its behavior with other aircraft. In support of the various missions, TacAir-Soar will need to know how to fly as many as 50 different aircraft types. Although many of these aircraft are similar to the aircraft it can already fly—at least at the level they are modeled for our simulations—some, such as the helicopters, are quite different. We anticipate that these new types of mission and aircraft will drive much of our future research and development effort in this area.

Although the requirements with regard to scope are thus at least somewhat clear, the requirements with regard to competence are not. In particular, the goal of the TacAir-Soar system is not to win a simulated air combat by any and all means (including simulator quirks). Rather, the goal involves remaining faithful to the specified constraints, including the mission parameters and the continued display of humanlike behavior. Thus, in some scenarios, TacAir-Soar might fail at performing its mission in a tactical sense but still be completely successful in terms of the requirements for intelligent automated forces. Our plan, therefore, is to use constant feedback from, and comparisons with, expert human pilots to drive both the development and the evaluation of TacAir-Soar's competence (complemented with more abstract evaluations of individual system components, where feasible). Generally, this plan involves having experts watch simulated engagements involving TacAir-Soar and human pilots and then asking the experts to analyze and comment on the behavior of the participants. So far, we have received enthusiastic comments about TacAir-Soar's behavior, along with some specific suggestions about how its behavior is inappropriate. However, there is still a long way to go.

Related Work

There are at least four classes of systems to which TacAir-Soar needs to be compared. The first class includes systems that provide intelligent forces for battlefield simulations in general and air-combat simulations in particular. Semiautomated forces tend to be developed as combinations of finite-state machines (FSMs) and arbitrary code (Calder et al. 1993). This strategy has proven adequate for encoding fragments of well-defined behavior but has so far fallen considerably short of autonomously competent behavior (thus requiring frequent intervention by knowledgeable humans). FSM languages are simply too restrictive to support the representation of humanlike intelligence. The ability to use arbitrary code does provide significant improvements in flexibility; however, this flexibility of language unfortunately does not by itself translate into flexibility of behavior by the system that is programmed within the language. What is missing is the higher-level support necessary to enable such behavioral flexibility, such as the support for dynamic integration of knowledge during behavior and goal-driven problem solving and planning.

Most of the systems that go beyond the simple FSM strategy are structured as simple rule-based expert systems for tactical decision making (Kornell 1987; Zytkow and Erickson 1987). Given TacAir-Soar's own knowledge-intensive (rule-based) approach to decision making, it does share some similarities with such systems. However, although expert systems have some of the strengths required for realistic simulation, they are weak in other areas. For example, in a standard rule-based approach, it is difficult to capture the complexity of the multiple dynamic goals about which pilots must reason. In addition, these systems typically rely only on high-level tactical knowledge and, as a result, prove to be rather rigid—unless they can be preprogrammed for every possible contingency,

Soar

All tasks in Soar are formulated as attempts to achieve goals in problem spaces (Newell et al. 1991). Each problem space consists of a set of states and a set of operators. States represent situations, and operators represent actions. Operators perform the basic deliberate acts of the system. They can perform simple, primitive actions that modify the internal state (such as determining if all commit criteria are achieved) and/or generate primitive external actions (such as switching radar modes), or they can perform arbitrarily complex actions, such as executing a mission. The basic processing cycle is to repeatedly propose, select, and apply operators of the problem space to a state, moving ahead one decision at a time. Operator selection, application, and termination are all dynamically determined by the system's knowledge.

For expert-level performance, sufficient knowledge is generally available so that the selection of the next appropriate operator is not problematic. However, when operator-selection knowledge is insufficient to determine the next operator to apply, an impasse occurs, leading to the creation of a subgoal to determine which operator should be selected (possibly through some search or planning technique [Rosenbloom, Lee, and Unruh 1993; Laird, Newell, and Rosenbloom 1987]). Similarly, if an operator is too complex or unfamiliar for the available application knowledge to handle directly, an impasse occurs, leading to the creation of a subgoal to apply the operator. This type of operator and associated subgoal is ubiquitous in TacAir-Soar, where operators such as intercept are selected and, in turn, lead to a goal in which operators specific to intercepts, such as plan-intercept-geometry or employ-missile, are proposed, selected, and applied to carry out the intercept. This is a goal-decomposition scheme in which the decomposition of a goal is not static but is determined step by step as operators are dynamically selected and applied. Thus, subgoals arise during the process of selecting and applying operators, leading to the dynamic generation of a goal hierarchy. These subgoals disappear when the associated impasse is resolved, either because an operator can be selected or because a selected operator is terminated.

Each problem space for each goal in the hierarchy has its own state. Each such state includes a representation of all its supergoals (and their states)—Soar's subgoals are functionally at the metalevel (Rosenbloom, Laird, and Newell 1988)—plus, possibly, representations that are local to the particular goal and problem space. The top state includes all sensor data from the external environment, which is thus also available in all subgoals. At any time, states at any level of the goal hierarchy can change, usually through the changing of sensor values. Because Soar's knowledge is also active for all levels, an operator can terminate at any level of the hierarchy—including intermediate levels—at any time. Such a termination automatically flushes all lower levels of the hierarchy and can lead to the selection of a new operator to replace the terminated one.

All Soar's knowledge, be it for selection, application, or termination of operators, is stored in the form of productions (condition-action rules). Any changes in goals, states, and perceptions can cause these productions to fire. There is no conflict resolution, so all applicable productions are allowed to fire in parallel. *Operator-selection knowledge* consists of productions that test the current goal or state and generate symbolic preferences about the absolute or relative worth of operators. For example, given a particular situation, a production might generate a preference that an operator to turn left is better than an operator to turn right. As the state changes, some preferences can be retracted (if the situation no longer matches the conditions of the rules that generated them) and others generated, so that decisions—which are made by a fixed, architectural decision procedure—can always be based on preferences relevant to the current situation. *Operator-application knowledge* consists of productions that modify the state in accordance with the particular operator selected and, possibly, generate output commands. *Operator-termination knowledge* consists of productions that test the state and generate a termination signal if the state corresponds to what the operator is to accomplish.

Goals and their results form the basis of Soar's learning mechanism, *chunking*. Chunking acquires new productions, called *chunks*, that summarize the processing that leads to subgoal results, that is, to elements generated in subgoals that are accessible in supergoals. A chunk's actions are based on these results. Its conditions are based on those aspects of the supergoals that are relevant to the determination of the results. Once a chunk is learned, it can fire in relevantly similar future situations, directly producing the required result and possibly avoiding the impasse that led to its formation. This chunking process is a form of explanation-based learning (Mitchell, Keller, and Kedar-Cabelli 1986; Rosenbloom and Laird 1986).

their performance degrades greatly when faced with unexpected situations. Finally, expert systems generally ignore the other cognitive aspects of the task, such as modeling of other agents in the environment and behaving in a humanlike manner. Soar was specifically constructed to overcome such limitations of standard rule-based expert systems. Besides Soar, one other system in this class that attempts to provide for flexible behavior is a recent effort at the Australian Artificial Intelligence Institute (Rao et al. 1993) that attempts to use the procedural reasoning system (PRS) (Georgeff and Lansky 1986) as a basis for modeling pilots in air-combat simulation. However, this effort does not yet appear to be far enough along to evaluate the extent to which it can actually provide such flexibility in this demanding domain.

The second class of systems includes those based on other AI agent architectures that attempt the integration of a variety of component capabilities. This category of systems is much broader. For example, consider the large number of systems presented at the 1991, 1994, and 1995 AAAI Spring Symposium on Agents and Agent Architectures. Unfortunately, the characteristics of the space of agent designs and architectures are not well enough understood to enable a clear understanding of the relationship between TacAir-Soar and these other systems. About the best we can do at this point is to compare the number and types of capabilities exhibited by these systems. Based on such an evaluation, TacAir-Soar appears to be one of a very few systems that integrate so many distinct capabilities (although clearly not all its capabilities exist in as general a form as they ultimately ought to). Another strong effort in the arena of highly integrated agents would appear to be Vere and Bickmore's (1990) BASIC AGENT, which combines limited natural language understanding and generation, planning, temporal reasoning, plan execution, episodic memory, simulated symbolic perception, and some general world knowledge.

The third class is other systems implemented within the Soar architecture. Although there are many such systems (descriptions of a number of them can be found in Rosenbloom, Laird, and Newell [1993]), two important characteristics of TacAir-Soar set it apart. First, TacAir-Soar is to actually be fielded and used in a real-world environment. This simulation environment is real world in the sense that it has been developed by outside (commercial and government) groups, it is of significant direct value to these groups, and they have a direct need for participation in it by intelligent agents. Second, TacAir-Soar requires the integration of a more diverse set of capabilities than any previous Soar-based system. Most of these capabilities have already been exhibited by individual Soar-based systems—knowledge-intensive and goal-driven behavior (Prietula et al. 1991; Rosenbloom et al. 1985), reactivity and real-time performance (Pearson et al. 1993; Laird and Rosenbloom 1990), planning (Rosenbloom, Lee, and Unruh 1993; Stobie, Tambe, and Rosenbloom 1992), agent modeling (Hill and Johnson 1994; Ward 1991), and natural language communication (Lehman, Lewis, and Newell 1991)—and these earlier systems have provided the strong foundations necessary for the work on TacAir-Soar. However, most of these systems, in reflecting the research interests of their developers, have targeted individual capabilities in isolation or a small number of capabilities in combination. There have been initial promising steps toward integration of multiple capabilities (for example, Huffman [1994]; Nelson, Lehman, and John [1994]; Laird and Rosenbloom [1990]; Polk, Lewis, and Newell [1989]), but these have been exceptions. We have thus realized for some time now, as have some others, a need for Soar systems that provide a tighter integration of a wider range of capabilities. For example, Hayes-Roth (1993) in her comments on the work on integration in Soar states,

> Let me presume to offer a friendly challenge to the Soar team. Give us "Agent-Soar," integrating all of the "content" of the existing Soar applications in a graceful, principled fashion within a single instance of the Soar architecture. Show us the resulting economies of storage and Agent-Soar's global perspective on its knowledge. Show us how Agent-Soar can exploit all of its content to meet several of the component requirements for intelligence simultaneously.

TacAir-Soar takes some concrete steps to address this friendly challenge. As outlined in the previous section, TacAir-Soar already integrates a diverse set of capabilities, and it promises to integrate even more, including natural language and learning.

The fourth class is systems that specialize in TacAir-Soar's individual capabilities, such as planning or agent modeling. These capabilities represent core research issues in AI, which our project must address, although within the confines of creating integrated pilot agents. We necessarily look to these spe-

cialized systems to aid us in this endeavor. However, there are just too many of them to even try to list them here.

Conclusion

Interactive simulation environments provide rich virtual worlds for research in building integrated intelligent agents that can interact with humans, each other, and their environment. Our current project is based on one such virtual world: battlefield simulations. We are building automated pilot agents that can participate in simulated combat, both with and against humans. The challenge this task poses is the creation of intelligent pilot agents that integrate a broad range of capabilities in a humanlike manner.

We are building a system called TacAir-Soar to address this challenge. Given the support it obtains from the Soar architecture, TacAir-Soar is easily able to exhibit a number of the required capabilities—to the extent that it is able to be the first AI system to directly participate in an operational military exercise. However, there are other capabilities for which the Soar architecture provides only a framework for their development or indeed proves somewhat of a hindrance to their development. These individual capabilities—and their integration in TacAir-Soar—have presented an interesting set of issues for both current and future research. They will also likely contribute to further evolution of the Soar architecture.

A second important set of issues for future work concerns the evaluation of the agents. Clearly, one important and large-scale test for the agents is the STOW demonstration in 1997. Between 10,000 and 50,000 automated agents and as many as 1,000 humans are to participate in this exercise. This demonstration will provide a gross evaluation of the viability, realism, and usefulness of our agents. However, additional forms of evaluation are also required. These include a detailed analysis of agent performance in terms of the individual capabilities and components and the interactions resulting from their integration as well as detailed comparisons of the behavior of the agents to humans and debriefings of agents by human subject-matter experts. Lessons learned from such evaluations could help extend this intelligent-agent technology beyond battlefield simulations into other collaborative and competitive virtual worlds for education, entertainment, training, and manufacturing.

Acknowledgments

We gratefully acknowledge the contribution of the other members of the team involved in the creation of the TacAir-Soar system, particularly Jill Lehman, Paul Nielsen, and Robert Rubinoff.

This research was supported under contract N00014-92-K-2015 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and from the Naval Research Laboratory (NRL) to the University of Michigan (and a subcontract to the University of Southern California/Information Sciences Institute from the University of Michigan). This project would not have been possible without the visionary support of Commander Dennis McBride of ARPA/ASTO and the technical support of Ed Harvey, Tom Brandt, Bob Richards, and Craig Petersen of BMH Inc.; Andy Ceranowicz and Joshua Smith of Loral Inc.; and David Keirsey of Hughes Aircraft Co.

Notes

1. ModSAF also provides other types of air, ground, and sea vehicles not discussed here as well as facilities for creating and controlling semiautomated forces.

2. LL's radar has other modes of operation, not all of them based on the Doppler principle. However, the Doppler mode is necessary for long-range missile firing.

References

Agre, P. E., and Chapman, D. 1987. PENGI: An Implementation of a Theory of Activity. In Proceedings of the Sixth National Conference on Artificial Intelligence, 268–272. Menlo Park, Calif.: American Association for Artificial Intelligence.

Anderson, J. R.; Boyle, C. F.; Corbett, A. T.; and Lewis, M. W. 1990. Cognitive Modeling and Intelligent Tutoring. *Artificial Intelligence* 42:7–49.

Bates, J.; Loyall, A. B.; and Reilly, W. S. 1992. Integrating Reactivity, Goals, and Emotions in a Broad Agent, Technical Report, CMU-CS-92-142, School of Computer Science, Carnegie Mellon University.

Bratman, M. E.; Israel, D. J.; and Pollack, M. E. 1988. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence* 4(4): 349–355.

Brooks, R. A. 1991. Intelligence without Representation. *Artificial Intelligence* 47:139–160.

Calder, R. B.; Smith, J. E.; Courtemanche, A. J.; Mar, J. M. F.; and Ceranowicz, A. Z. 1993. ModSAF Behavior Simulation and Control. In Proceedings of the Third Conference on Computer-Generated Forces and Behavioral Representation, 347–356. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Covrigaru, A. 1992. Emergence of Meta-Level Control in Multi-Tasking Autonomous Agents. Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of Michigan.

Cremer, J.; Kearney, J.; Papelis, Y.; and Romano, R.

1994. The Software Architecture for Scenario Control in the Iowa Driving Simulator. In Proceedings of the Fourth Conference on Computer-Generated Forces and Behavioral Representation, 373–381. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Crooks, W. H.; Fraser, R. E.; Jacobs, R. S.; and McDonough, J. G. 1992. Functional Description: WISSARD Aircraft Simulators, Technical Report TR-11100-15600-08-92, Illusion Engineering Inc., Westlake Village, California.

DIS Steering Committee. 1994. The DIS Vision: A Map to the Future of Distributed Simulation, Technical Report, IST-SP-94-01, Institute for Simulation and Training, University of Central Florida.

Durfee, E., and Montgomery, T. A. 1990. A Hierarchical Protocol for Coordinating Multi-Agent Behavior. In Proceedings of the Eighth National Conference on Artificial Intelligence, 86–93. Menlo Park, Calif.: American Association for Artificial Intelligence.

Etzioni, O. 1993. Intelligence without Robots: A Reply to Brooks. *AI Magazine* 14(4): 7–13.

Georgeff, M. P., and Lansky, A. L. 1986. Procedural Knowledge. In *Proceedings of the IEEE* (Special Issue on Knowledge Representation) 74:1383–1398.

Gunston, B. 1986. *AH-64 APACHE*. London: Osprey.

Hanks, S.; Pollack, M. E.; and Cohen, P. R. 1993. Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures. *AI Magazine* 14(4): 17–42.

Hayes-Roth, B. 1993. On Building Integrated Cognitive Agents: A Review of Allen Newell's Unified Theories of Cognition. *Artificial Intelligence* 59:329–341.

Hill, R., and Johnson, W. L. 1994. Situated Plan Attribution for Intelligent Tutoring. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 499–505. Menlo Park, Calif.: American Association for Artificial Intelligence.

Huffman, S. B. 1994. Instructable Autonomous Agents. Ph.D. thesis, Department of Computer Science and Engineering, University of Michigan.

IEEE Standards Board. 1993. IEEE Standard for Information Technology—Protocols for Distributed Interactive Simulation Applications, Technical Report IEEE-Std-1278-1993, Institute for Electrical and Electronics Engineers, Washington, D.C.

Johnson, W. L. 1994a. Agents That Explain Their Own Actions. In Proceedings of the Fourth Conference on Computer-Generated Forces and Behavioral Representation, 87–95. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Johnson, W. L. 1994b. Agents That Learn to Explain Themselves. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 1257–1263. Menlo Park, Calif.: American Association for Artificial Intelligence.

Johnson, W. L.; Jones, R. M.; Keirsey, D.; Koss, F. V.; Laird, J. E.; Lehman, J. F.; Neilsen, P. E.; Rosenbloom, P. S.; Rubinoff, R.; Schwamb, K.; Tambe, M.; van Lent, M.; and Wray, R. 1994. Collected Papers of the Soar/IFOR Project, Spring 1994, Technical Report, CSE-TR-207-94, Department of Electrical Engineering and Computer Science, University of Michigan.

Jones, R. M. 1994. Dynamic Generation of Complex Behavior. Video Presented at the Twelfth National Conference on Artificial Intelligence, 31 July–5 August, Seattle, Washington.

Jones, R.; 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, Fla.: University of Central Florida Institute for Simulation and Training.

Jones, R. M.; Tambe, M.; Laird, J. E.; and Rosenbloom, P. 1993. Intelligent Automated Agents for Flight Training Simulators. In Proceedings of the Third Conference on Computer-Generated Forces and Behavioral Representation, 33–42. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Kautz, A., and Allen J. F. 1986. Generalized Plan Recognition. In Proceedings of the Fifth National Conference on Artificial Intelligence, 32–37. Menlo Park, Calif.: American Association for Artificial Intelligence.

Kornell, J. 1987. Reflections on Using Knowledge-Based Systems for Military Simulation. *Simulation* 48:144–148.

Laird, J. E., and Rosenbloom, P. S. 1990. Integrating Execution, Planning, and Learning in Soar for External Environments. In Proceedings of the Eighth National Conference on Artificial Intelligence, 1022–1029. Menlo Park, Calif.: American Association for Artificial Intelligence.

Laird, J. E.; Jones, R. M.; and Nielsen, P. E. 1994. Coordinated Behavior of Computer-Generated Forces in TacAir-Soar. In Proceedings of the Fourth Conference on Computer-Generated Forces and Behavioral Representation, 325–332. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An Architecture for General Intelligence. *Artificial Intelligence* 33(1): 1–64.

Laird, J. E.; Congdon, C. B.; Altmann, E.; and Doorenbos, R. 1993. Soar User's Manual, version 6. Available from soar-doc@isi.edu.

Lehman, J. F.; Lewis, R. L.; and Newell, A. 1991. Natural Language Comprehension in Soar: Spring 1991, Technical Report, CMU-CS-91-117, School of Computer Science, Carnegie Mellon University.

Maes, P.; Darrell, T.; Blumberg, B.; and Pentland, S. 1994. Interacting with Animated Autonomous Agents. Presented at the AAAI Spring Symposium on Believable Agents, 21–23 March, Stanford, California.

Mitchell, T. M.; Keller R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-Based Generalization: A Unifying View. *Machine Learning* 1(1): 47–80.

Moravec, H. 1990. *Mind Children*. Cambridge, Mass.: Harvard University Press.

Nelson, G.; Lehman, J. F.; and John, B. E. 1994. Integrating Cognitive Capabilities in a Real-Time Task. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society.* Hillsdale, N.J.: Lawrence Erlbaum.

Newell, A. 1992a. Precis of Unified Theories of Cognition. *Behavioral and Brain Sciences* 15:425–492.

Newell, A. 1992b. Unified Theories of Cognition and the Role of Soar. In *Soar: A Cognitive Architecture in Perspective,* eds. J. Michon and A. Akyurek. Cambridge, Mass.: Kluwer Academic.

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

Newell, A.; Yost, G. R.; Laird, J. E.; Rosenbloom, P. S.; and Altmann, E. 1991. Formulating the Problem Space Computational Model. In *CMU Computer Science: A 25th Anniversary Commemorative,* ed. R. F. Rashid, 255–293. New York: ACM Press/Addison-Wesley.

Pearson, D. J.; Huffman, S. B.; Willis, M. B.; Laird, J. E.; and Jones, R. M. 1993. A Symbolic Solution to Intelligent Real-Time Control. *IEEE Robotics and Autonomous Systems* 11:279–291.

Polk, T. A.; Newell, A.; and Lewis, R. L. 1989. Toward a Unified Theory of Immediate Reasoning in Soar. In *Proceedings of the Annual Conference of the Cognitive Science Society,* 506–513. Hillsdale, N.J.: Lawrence Erlbaum.

Prietula, M.; Hsu, W-L.; Steier, D.; and Newell, A. 1991. Applying an Architecture for General Intelligence to Scheduling. *ORSA Journal on Computing* 5(3).

Rao, A. S., and Murray, G. 1994. Multi-Agent Mental-State Recognition and Its Application to Air-Combat Modelling. In Proceedings of the Workshop on Distributed Artificial Intelligence (DAI-94). Menlo Park, Calif.: American Association for Artificial Intelligence.

Rao, A. S.; Lucas, A.; Morley, D.; Selvestrel, M.; and Murray, G. 1993. Agent-Oriented Architecture for Air-Combat Simulation, Technical Note 42, The Australian Artificial Intelligence Institute.

Rosenbloom, P. S., and Laird, J. E. 1994. On Unified Theories of Cognition: A Response to the Reviews. In *Contemplating Minds: A Forum for Artificial Intelligence,* eds. W. J. Clancey, S. W. Smoliar, and M. J. Stefik, 141–165. Cambridge, Mass.: MIT Press.

Rosenbloom, P. S., and Laird, J. E. 1986. Mapping Explanation-Based Generalization onto Soar. In Proceedings of the Fifth National Conference on Artificial Intelligence, 561–567. Menlo Park, Calif.: American Association for Artificial Intelligence.

Rosenbloom, P. S.; Laird, J. E.; and Newell, A., eds. 1993. *The Soar Papers: Research on Integrated Intelligence.* Cambridge, Mass.: MIT Press.

Rosenbloom, P. S.; Laird, J. E.; and Newell, A. 1988. Meta-Levels in Soar. In *Meta-Level Architectures and Reflection,* eds. P. Maes and D. Nardi, 227–240. Amsterdam: North Holland.

Rosenbloom, P. S.; Laird, J. E.; and Newell, A. 1987. Knowledge Level Learning in Soar. In Proceedings of the Sixth National Conference on Artificial Intelligence, 499–504. Menlo Park, Calif.: American Association for Artificial Intelligence.

Rosenbloom, P. S.; Laird, J. E.; Newell, A.; and McCarl, R. 1991. A Preliminary Analysis of the Soar Architecture as a Basis for General Intelligence. *Artificial Intelligence* 47(1–3): 289–325.

Rosenbloom, P. S.: Laird, J. E.; McDermott, J.; Newell, A.; and Orciuch, E. 1985. R1-Soar: An Experiment in Knowledge-Intensive Programming in a Problem-Solving Architecture. *Pattern Analysis and Machine Intelligence* 7:561–569.

Rosenbloom, P. S.; Lee, S.; and Unruh, A. 1993. Bias in Planning and Explanation-Based Learning. In *Machine Learning Methods for Planning,* ed. S. Minton, 197–232. San Francisco, Calif.: Morgan Kaufmann.

Rubinoff, R., and Lehman, J. 1994. Natural Language Processing in an IFOR Pilot. In Proceedings of the Fourth Conference on Computer-Generated Forces and Behavioral Representation, 97–105. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Schwamb, K.; Koss, F.; 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, 395–400. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Stobie, I.; Tambe, M.; and Rosenbloom, P. 1992. Flexible Integration of Path-Planning Capabilities. In Proceedings of the SPIE Conference on Mobile Robots, 15–20 November, Boston, Massachusetts.

Tambe, M.; Jones, R. M.; Laird, J. E.; and Rosenbloom, P. S. 1994. Building Believable Agents for Simulation Environments. Presented at the AAAI Spring Symposium on Believable Agents, 21–23 March, Stanford, California.

Tambe M., and Rosenbloom, P. S. 1994. Event Tracking in Complex Multi-Agent Environments. In Proceedings of the Fourth Conference on Computer-Generated Forces and Behavioral Representation, 473–485. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Thorpe, J. A.; Shiflett, J. E.; Bloedorn, G. W.; Hayes, M. F.; Miller, D. C. 1989. The SIMNET Network and Protocols, Technical Report 7102, BBN Systems and Technologies Corporation.

van Lent, M., and Wray, B. 1994. A Very Low Cost System for Direct Human Control of Simulated Vehicles. In Proceedings of the Fourth Conference on Computer-Generated Forces and Behavioral Representation, 79–86. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Vere, S., and Bickmore, T. 1990. A BASIC AGENT. *Computational Intelligence* 6:41–59.

Ward, B. 1991. ET-Soar: Toward an ITS for Theory-Based Representations. Ph.D. thesis, School of Computer Science, Carnegie Mellon University.

Webber, B., and Badler, N. 1993. Virtual Interactive Collaborators for Simulation and Training. In Proceedings of the Third Conference on Computer-Generated Forces and Behavioral Representation,

199–207. Orlando, Fla.: University of Central Florida Institute for Simulation and Training.

Zytkow, J. M., and Erickson, M. D. 1987. *Tactical Manager in a Simulated Environment. Methodologies for Intelligent Systems.* Amsterdam: Elsevier Science.

Milind Tambe is a research computer scientist at the University of Southern California Information Sciences Institute, and a research assistant professor in the USC Department of Computer Science. He completed his undergraduate education in computer science from the Birla Institute of Technology and Science, Pilani, India in 1986, and received his Ph.D. in 1991 in Computer Science from Carnegie Mellon University. His research interests include integrated intelligent agents, and real-time behavior, parallelism and scalability of Al systems, especially production (rule-based) systems.

W. Lewis Johnson is a project leader at the University of Southern California Information Sciences Institute, and a research assistant professor in the USC Department of Computer Science. Johnson received his A.B. degree in Linguistics in 1978 from Princeton University, and his M.Phil. and Ph.D. degrees in Computer Science from Yale University in 1980 and 1985, respectively. He is interested in applying artificial intelligence techniques in the areas of computer-based training and software engineering. His current projects are developing tools that automate the generation of software documentation, and that explain the problem solving of intelligent agents.

Randolph M. Jones received his B.S. in Mathematics and Computer Science from the University of California, Los Angeles, in 1984. In 1987 and 1989, respectively, he received M.S. and Ph.D. degrees in Information and Computer Science from the University of California, Irvine. He next served as a research associate for a year in the Psychology Department at Carnegie Mellon University, and for two years at the Learning Research and Development Center of the University of Pittsburgh. He is currently an assistant research scientist in the Artificial Intelligence Laboratory at the University of Michigan. His research interests lie in the areas of intelligent agents, problem solving, machine learning, analogical reasoning, and psychological modeling.

Frank V. Koss is a systems research programmer in the Artificial Intelligence Laboratory at the University of Michigan, where he is developing the interface between the Soar cognitive architecture and the ModSAF simulator and extending ModSAF itself. He received his BS in computer engineering from Carnegie Mellon University in 1991 and his MSE in computer science and engineering from the University of Michigan in 1993.

John E. Laird is an associate professor of electrical engineering and computer science and the director of the Artificial Intelligence Laboratory at the University of Michigan. He received his B.S. degree in computer and communication sciences from the University of Michigan in 1975 and his M.S. and Ph.D. degrees in computer science from Carnegie Mellon University in 1978 and 1983, respectively. His interests are centered on creating integrated intelligent agents (using the Soar architecture), leading to research in problem solving, complex behavior representation, machine learning, and cognitive modeling.

Paul S. Rosenbloom is an associate professor of computer science at the University of Southern California and the acting deputy director of the Intelligent Systems Division at the USC Information Sciences Institute. He received his B.S. degree in mathematical sciences from Stanford University in 1976 and his M.S. and Ph.D. degrees in computer science from Carnegie-Mellon University in 1978 and 1983, respectively. His research centers on integrated models of intelligent behavior in both natural and artificial systems (and, in particular, on Soar), but also covers other areas such as machine learning, production systems, planning, and cognitive modeling. He is a Councillor and Fellow of the AAAI and a past Chair of ACM SIGART.

Karl Schwamb is a Programmer Analyst on the Soar Intelligent Forces project at the University of Southern California Information Sciences Institute. He maintains the Soar/ModSAF interface software and enhances the Soar infrastructure. He received his M.S. in Computer Science from George Washington University.