Constraints and Design Choices in Building Intelligent Pilots for Simulated Aircraft: Extended Abstract

Milind Tambe, Paul S. Rosenbloom and Karl Schwamb
Information Sciences Institute
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
email: {tambe, rosenbloom, schwamb}@isi.edu

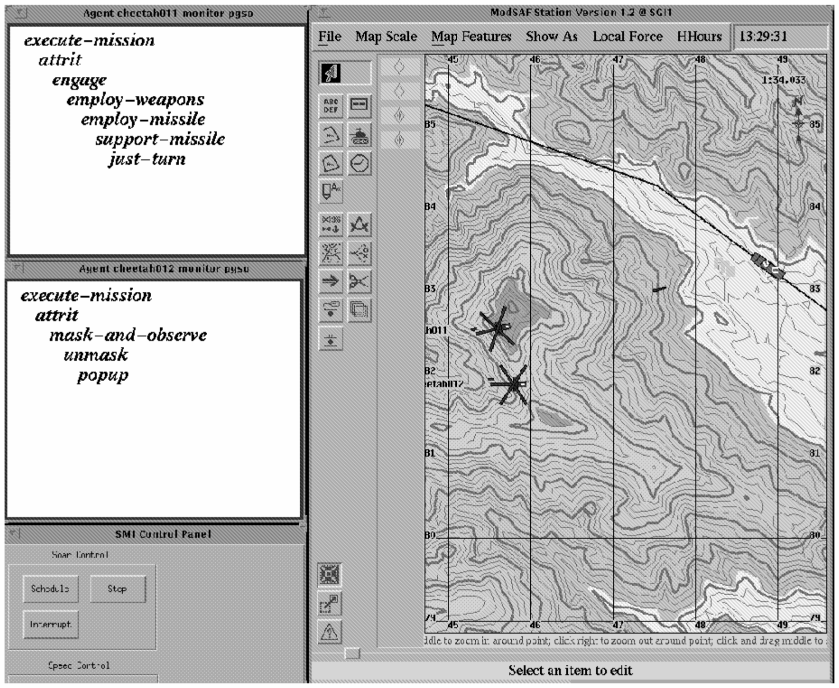
1. Introduction

This paper focuses on our recent research effort aimed at developing human-like, intelligent agents (*virtual humans*) for large-scale, interactive simulation environments (*virtual reality*). These simulated environments have sufficiently high fidelity and realism [11, 23] that constructing intelligent agents requires us to face many of the hard research challenges faced by physical agents in the real world — in particular, the integration of a variety of intelligent capabilities, including goal-driven behavior, reactivity, real-time performance, planning, learning, spatial and temporal reasoning, and natural language communication. However, since this is a synthetic environment, these intelligent agents do not have to deal with issues of low-level perception and robotic control. Important applications of this agent technology can be found in areas such as education [14], manufacturing [11], entertainment [2, 12] and training [7, 24].

To begin our effort, we have focused on building intelligent pilot agents to control simulated military aircraft in battlefield simulation environments. These environments are based on Distributed Interactive Simulation (DIS) technology [11], in which large-scale interactive simulations, potentially involving hundreds or even thousands of simulated entities, are built from a set of independent simulators linked together via a network. These simulations provide cost-effective and realistic settings for training and rehearsal, as well as testing new doctrine, tactics and weapon system concepts. Our intelligent pilot (henceforth IP) agents are to engage in simulated combat with other automated agents and humans in these environments. The IPs interact with their environments via simulated aircraft provided by ModSAF [4], a distributed simulator that has been commercially developed for the military.

Most of our early research effort, since the beginning of this project in the Fall of 1992, was focused on the creation of IPs for simulated fighter jets. These IPs, either individually or in groups, can now engage in simulated "air-to-air" combat with other individual or groups of fighters.[1] Since the summer of 1994, we have begun developing IPs for other types of aircraft, such as bombers and attack helicopters, involved in very different types of missions, such as "air-to-ground" missions. Figure 1 shows a snapshot taken from ModSAF's plan-view display of a combat situation. Here, IPs are controlling simulated AH-64 Apache attack helicopters. The other vehicles in the figure are a convoy of "enemy" ground vehicles such as tanks, and anti-aircraft vehicles. The helicopters are at approximately 2.5 miles from the convoy of ground vehicles. (All of the vehicle icons in the figure have been artificially magnified for visibility.) The background shading delineates the terrain and contour lines. The contour lines indicate that the two helicopters are on a hill, just behind a ridge, while the convoy of enemy vehicles is in a valley. This is an ideal location for the IPs to hide their helicopters. The IPs unmask their helicopters by popping out from behind the ridge to launch missiles at the enemy vehicles, and quickly remask (hide) by dipping behind the ridge to survive retaliatory attacks. The smaller windows along the side of the figure display information about the active goal hierarchies of the IPs controlling the helicopters. In this case, one of the IPs has seen the enemy vehicles, and is firing a missile at them. The second IP is unmasking its helicopter. (The small window at the bottom left is for simulation control.) The eventual target of our effort is to deliver to ARPA IPs for a large variety of simulated military aircraft, including fighter jets, helicopters, bombers, troop transports, refueling aircraft and reconnaissance aircraft. These IPs are to participate in a series of demonstrations called the simulated theater of war (STOW) demonstrations. One such demonstration, called STOW-E, took place in November 1994. It involved about two thousand simulated military vehicles — some of them human controlled and others by automated agents. IPs participated successfully in this demonstration. The next demonstration, in 1997, is planned to be a much larger scale one, involving the participation of ten to fifty thousand automated agents, and up to one thousand humans.

[1]The work on fighter jet IPs was a collaborative effort with Randolph Jones, John Laird, Frank Koss and Paul Nielsen of the University of Michigan.



```
execute-mission
  attrit
    engage
      employ-weapons
        employ-missile
          support-missile
            just-turn
```

```
execute-mission
  attrit
    mask-and-observe
      unmask
        popup
```

Figure 1: A snapshot of ModSAF's simulation of an air-to-ground combat situation.

For research and development of the IPs, we have focused on Soar, a software architecture being developed as a basis for integrated intelligent systems [10, 18], and as a unified theory of cognition [15, 17]. (For readers unfamiliar with Soar, a brief description appears in the Appendix.) The system we have developed within Soar, TacAir-Soar [6, 22], represents a generic IP. A specific IP is created by specializing it with specific parameters and domain knowledge. TacAir-Soar-based IPs can fly fighter jets, bombers and attack helicopters, on different types of "aircraft vs aircraft", as well as "aircraft vs ground target" combat missions. The creation of these IPs has consumed about two year's worth of effort of our research team of about six people.

This paper outlines some of the constraints and design choices in our effort to implement IPs in the form of TacAir-Soar. Overall, the design of TacAir-Soar IPs is guided by two general sets of constraints. The first set of constraints, which we will refer to as *top-down* constraints, arises from the combat simulation environment. Figure 2 lists these constraints in the form of capabilities that an IP must possess if it is to succeed in this environment. We have attempted to match these capabilities with the list of issues provided by the symposium organizers in their call for papers. The match is not one-to-one, and in some cases, there are multiple capabilities that are relevant to an issue. Two of the issues — agent generality and awareness of space and time — are our additions to the list of issues. Note that this simulation environment has been commercially developed based on the specifications of the military, and as agent designers, we have little control over it. Thus, the list of capabilities cannot be modified to suit our (agent designer's) needs.

Issue 1::*Knowledge representation and organization*: Knowledge must be organized and represented such that IPs possess the following capabilities:

a. *Goal-driven behavior*: IPs need to accomplish the missions specified to them, while remaining faithful to the mission-related constraints. In Figure 1, the IP's mission is to provide support to friendly ground forces while being stationed behind a hill. These missions and mission related parameters help to specify a pilot's goals, including high-level goals such as execute mission and survive, and lower level goals such as fire a missile at an opponent, hide from an opponent to evade his/her missile etc.

b. *Knowledge-intensive behavior*: For an IP to be able to generate goal-driven behavior, it needs a large amount of knowledge related to the various missions, mission-related parameters, tactics, maneuvers, aircraft, missile and radar types and their parameters, etc.

Issue 2::*Coordination of actions and behaviors:* In addition to the capability of goal-driven behavior mentioned above, the capabilities relevant to this issue are:

a. *Reactivity*: Situations in combat environments can change very rapidly. Unexpected events can occur, e.g., an on-target missile may fail to explode, or an aggressive adversary may take some preemptive action disrupting an on-going maneuver. An IP must be able to react to such a rapidly evolving situation.

b. *Overlapping of performance of multiple high-level tasks:* IPs often have to perform multiple tasks simultaneously, e.g., an IP may be increasing the altitude of its helicopter, while communicating with an IP of a friendly helicopter, and simultaneously attempting to identify the vehicles in its field of view.

c. *Planning*: A helicopter's IP may need to replan a route if it sees that the planned one is blocked; or it may need to plan a hiding location for its helicopter after launching missiles at the enemy as in Figure 1.

Issue 3::*Learning*: Although learning has not proven to be a critical requirement so far, we expect learning to eventually play an important role in improving performance both during and across engagements.

Issue 4::*Human-like behavior*: For realism in training and tactics development, an IP must exhibit human-like capabilities for goal-driven behavior, reactivity, planning, learning, and in fact all of the other capabilities listed in this figure. In addition, an IP must conform to human reaction times and other human limitation. Thus, an IP must not maneuver a simulated aircraft like either a "superhuman" or an idiot.

Issue 5::*Real-time Performance*: IPs have to participate in combat simulations where they may interact with humans acting in real-time.

Issue 6::*Generality*: Although our effort began with implementing IPs for simulated F-14 fighter aircraft in a specific air-to-air combat situation, the effort has extended to other combat situations, other fighter aircraft, and now to air-to-ground combat situations faced by helicopters. This has forced a continuous movement towards design generality, so that capabilities can be shared and re-used.

Issue 7::*Awareness of space and time*: The capabilities involved here include:

a. *Spatial reasoning*: Helicopter pilots need to reason about the terrain, e.g., to hide their helicopters behind a hill, so that they can protect themselves from enemy fire.

b. *Temporal reasoning*: IPs need to reason about the effect of actions over time intervals, e.g., to locate the position of an opponent who may have become invisible for a minute.

Issue 8::*Interface with other agents*: The capabilities involved include:

a. *Multi-agent co-ordination:* Mission specifications for IPs normally require them to execute missions in groups of two, four or eight aircraft. Coordination among such agents is key to the success of such missions.

b. *Agent modeling (especially opponent modeling):* For effective interaction with other agents, IPs need to track other agents' low-level actions, such as the movement of their aircraft, to infer higher-level plans, maneuvers and goals.

c. *Communication*: Communication is important for coordination among IPs, e.g., they may communicate in performing a coordinated maneuver or exchange information about enemy aircraft.

d. *Explanation*: An IP needs to explain the reasons for its actions to "domain experts", so that they may have some confidence in its decision-making capability.

Figure 2: List of capabilities for Intelligent Pilots (based on [22]).

The second set of constraints arises from the fact that in designing IPs, we have not engineered a new agent architecture from scratch. Instead, we have begun with Soar as the underlying architecture. Soar embodies a variety of constraints resulting from theoretical and psychological considerations, as well as refinements inspired by its application in a large number and variety of domains over the past dozen years. We will refer to these constraints as *bottom up* constraints. A majority of them are *positive* in that they provide concrete integrated solutions, so that TacAir-Soar-based IPs can easily exhibit many of the required capabilities. However, for some of the capabilities, the constraints provide only a loose framework, and in fact, in some cases they appear to be a hindrance. Should Soar be modified in response to such hindrances? The answer is not obvious as yet. Given that the design decisions in Soar attempt to capture key insights from a variety of domains, architectural modifications are not to be taken lightly. The hindrances that Soar offers could actually be opening up new avenues for addressing the relevant issues.

We may consider the TacAir-Soar IPs, although based on Soar and constrained by Soar, as nonetheless possessing an architecture that is distinct from Soar. This paper will focus on the constraints — both top-down and bottom-up — on this IP architecture, as well as the design choices involved in it. The next section discusses these constraints and design choices in more detail. In doing so, it focuses on the issues listed in Figure 2, particularly issues 1-6. Although issues 7-8 are also important, most of the relevant capabilities are currently under investigation, and their interactions with the Soar architecture are as yet unclear.

2. Issues in Designing Intelligent Pilots

We begin with a discussion of the knowledge organization and representation in TacAir-Soar, since that illustrates the basic design of the system. The subsequent sequencing of issues is an attempt to best illustrate various design choices in TacAir-Soar.

2.1. Knowledge Representation and Organization

The top-down environmental constraint on knowledge organization and representation is that TacAir-Soar IPs must have the capabilities for goal-driven and knowledge intensive behavior. The bottom-up constraint offered by Soar is in the form of its architectural primitives — specifically, goals, problem spaces, states and operators at an abstract problem space level, and a rule-based system plus a preference language at a lower implementation level — that aid in knowledge organization and representation (Appendix I presents a description of these levels). All of the knowledge in TacAir-Soar is represented in terms of these architectural primitives at the two levels. In this section, we will focus on the problem space level; we will return to the lower implementation level in the next section.

Given the above top-down and bottom-up constraints, the available design choice is to select the appropriate problem spaces — with appropriate combinations of operators — from the knowledge acquired from the domain experts. An appropriate problem space refers to the relevant knowledge being quickly available to an IP when it attempts to achieve a particular goal. For example, a problem space combining operators for employing different weapon types enables an IP to quickly access relevant knowledge to achieve its goal of destroying an opponent. If the same set of operators are spread out in different problem spaces, an IP would not be able to easily and quickly access them. As another example, consider a complex tactic employed by an IP to avoid enemy fire in Figure 1. It masks its helicopter by dipping behind the ridge, flies at a very low altitude to a different location while still masked behind the ridge (so the enemy cannot predict its location) and then unmasks by popping out from behind the ridge. Representing this tactic as a set of appropriately conditioned operators that mask, select a new masking location, move to new masking location and unmask, ensures that the knowledge is available in other situations where there may be a need to unmask. If the same tactic is represented as a single monolithic operator, it would be unavailable (this motivation is related to an additional motivation that big monolithic operators are difficult to execute in this dynamic environment).

To describe the basics the knowledge organization that results from the above consideration, we will focus on a concrete illustrative example of an IP as it controls a helicopter and fires a missile at a tank, as shown in Figure 1. Figure 3 depicts an IP's hierarchy of problem spaces and operators. Goals are represented implicitly in this diagram as the desires to apply impassed operators. Boxes indicate problem spaces, with names in bold to their right. Names within boxes indicate operators available within problem spaces. An italicized name indicates a currently selected operator in a problem space; non-italicized names indicate the other available operators in the problem space.

This hierarchy gets generated as follows. In the top-most problem space (**TOP-PS**), the IP is attempting to accomplish its mission by applying the *EXECUTE-MISSION* operator. The termination condition of this operator is the completion of the IP's mission (which is to support friendly ground forces for a mission-specified time period). Since this 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. Operators in this problem space — — ATTRIT, RAID, FLY-WING and others — implement

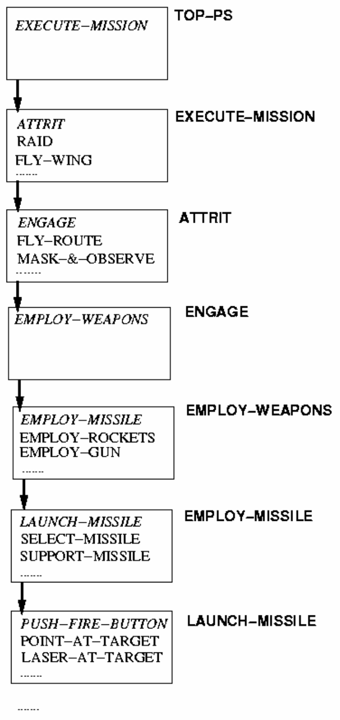


Figure 3: An IP's problem space hierarchy as it launch

the missions that IPs can engage in. IP selects the *ATTRIT* operator to perform its mission. The termination condition of this operator — that the opponent forces are reduced in strength — is also not yet achieved, leading to a second subgoal of completing this mission. The **ATTRIT** problem space is selected for use in this subgoal. Within this problem space the IP selects the *ENGAGE* operator, to engage the opponents in combat. This leads to a fourth subgoal, into the **ENGAGE** problem space, within which *EMPLOY-WEAPONS* is applied to either destroy the opponents or to force them to run away. This leads to a fifth subgoal into the **EMPLOY-WEAPONS** problem space. Here, the *EMPLOY-MISSILE* operator indicates the choice of a missile as the weapon to use in the given situation. This leads to the next subgoal. The **EMPLOY-MISSILE** problem space is selected for use in this subgoal. The *LAUNCH-MISSILE* operator then indicates that the IP is about to launch a missile. Launching a missile includes the operations of turning to point the missile launcher at an enemy vehicle, providing initial guidance to the missile (e.g., by illuminating the target with a laser), and also pushing the fire button. The *PUSH-FIRE-BUTTON* operator in the **LAUNCH-MISSILE** problem space performs this latter function, by issuing a fire missile command to the simulator, thus causing a missile to be fired at the selected target. If the firing of the missile causes the termination conditions of any of the operators in the hierarchy to be satisfied, then that operator is terminated, and all of the subgoals generated in response to that operator are removed — they are now irrelevant. In this case, since the firing of the missile causes the termination condition of the *LAUNCH-MISSILE* operator to be achieved, that operator is terminated and replaced by the *SUPPORT-MISSILE* operator, to provide remote guidance to the missile from the helicopter. The subgoals associated with the *LAUNCH-MISSILE* operator are flushed.

This goal/problem space hierarchy provides TacAir-Soar the capability of goal-driven behavior. In addition, it also provides the capability of knowledge-intensive behavior. The knowledge-intensive aspect arises because all of TacAir-Soar's behavior is driven by accessing relevant knowledge about action. The generation of alternative operators occurs via rules that examine the goal, problem space, and state and create instantiated operators that may 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.

Some evidence of the appropriateness of our initial problem space and operator design (from fighter jet IPs) came in the form of the extensive re-use of these problem spaces and operators for helicopter IPs. The air-to-air combat missions for fighter jet IPs and the air-to-ground combat missions for helicopter IPs are quite different. However, there are some skills that they do share, such as firing missiles; and indeed, this sharing is reflected in the reuse of problem spaces and operators from the fighter jet IPs in helicopter IPs. Table I illustrates this situation. It shows the total number of problem spaces, operators and rules in the helicopter IPs, the number reused from the fighter jet IPs, and the percentage reuse. Note that we are still in the early stages of developing a helicopter IP, and thus, the amount of re-use could change in the future.[2]

2.2. Coordination of Behaviors and Actions

The constraints here are as follows:

- *Top-down*: An IP must have the ability to combine goal-driven and reactive behaviors; it must also

[2]Since it is not straightforward to get a precise count of rule reuse, it is overestimated somewhat here. In particular, there are rules that are re-used from the fighter jet IPs, which although not useful for this task, are harmless, and have not been weeded out.

Entity reused	Total in Helicopter IP	Reused from Fighter jet IP	Percentage reuse
Problem spaces	14	6	42
Operators	45	29	64
Rules	1196	1030	86

Table 1: Reuse of problem spaces, operators and rules in IPs.

have the ability to simultaneously perform multiple high-level tasks. In addition, an IP must have the capability of planning, and constraining its behavior based on the resulting plans [3]. However, since we have only recently begun investigating the issue of planning in this environment, we will not discuss it here.

- *Bottom-up*: The relevant constraints arising from Soar are its decision procedure and its rule-based representation of knowledge, both of which facilitate TacAir-Soar's ability to support the above capabilities; and its single goal hierarchy, that possibly hinders these capabilities.

Here, there are two design decisions for TacAir-Soar IPs. The first, rather obvious one, is to take advantage of the features of the Soar architecture that facilitate the combination of goal-driven and reactive behaviors. These features are its open and integrative decision procedure and its production (rule-based) representation. All of TacAir-Soar's knowledge is encoded in terms of productions. These productions can test the current goals and operators being pursued (e.g., *EMPLOY-MISSILE* or *EVADE-MISSILE*), as well as relevant aspects of the current situation (e.g., sensors may indicate a sudden movement by an enemy vehicle). Based on either or both of these tests, the productions may suggest new operators to pursue at any level in the goal hierarchy, generate preferences among suggested operators, and terminate existing operators. 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. This enables TacAir-Soar-based IPs to be reactive, and also combine this reactivity with goal-driven behavior. While using rules in this manner to provide reactivity is similar to the approach taken in pure situated-action systems [1], there is one important difference: the use of persistent internal state. TacAir-Soar IPs cannot rely on the external environment to provide them all of the required cues for effective performance. For instance, even if an IP evades enemy fire by masking its helicopter behind a hill, it needs to maintain information regarding enemy position on its state. Not doing so would cause it to be repeatedly surprised by enemy fire when it unmasks, giving the enemy ample time to fire at the helicopter. Therefore, TacAir-Soar IPs do engage in a substantial effort to deliberately maintain internal state information.

With respect to simultaneous performance of multiple high-level tasks, there is an apparent hindrance from the Soar architecture — TacAir-Soar cannot directly construct multiple independent goal hierarchies in service of multiple high-level tasks. For instance, consider a situation where an IP needs to reason about a new vehicle in its field of view while simultaneously attempting to fire a missile. In this case, the IP constructs a goal hierarchy for firing the missile, as shown in Figure 3. The problem is that, though a second goal hierarchy is also required here — to identify the new vehicle in the field of view, to check if it is a threat to IP's helicopter, etc. — the IP cannot independently construct it because of the constraint imposed by Soar that there be only a single goal hierarchy.

There are two general approaches to resolving this apparent mismatch between Soar's architectural features and requirements of the environment [8]. The first approach is to view the single goal hierarchy as a negative constraint, and attempt to relax the constraint by proposing a change in the Soar architecture to allow the use of a forest of goal hierarchies. Such a forest would enable TacAir-Soar IPs to easily engage in simultaneous performance of multiple high-level tasks. However, changing the Soar architecture in this manner leads to important questions regarding the interactions of this approach with other aspects of the architecture. These questions are actively under investigation.

The second approach is to consider the mismatch caused by Soar's single goal hierarchy as only an apparent one rather than a real one, and attempt to develop solutions that will conform to this constraint. These include our currently implemented solution in TacAir-Soar. It requires an IP to commit to just one of the goal hierarchies, and to install operators for the remaining high-level goals, as needed, at the bottom of this goal hierarchy. Thus, in the above example, the IP commits to a hierarchy for firing a missile, but then installs an *identify-vehicle* operator as a subgoal of the *push-fire-button* operator. While this solution can be made to work, one significant problem here is that any attempt to update the upper goal hierarchy eliminates the lower one, because Soar automatically interprets "lower" as meaning "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 instant, however, a more general solution seems necessary. A second solution in this class is to attempt to merge operators across goal hierarchies so that a single hierarchy can represent the state of all of the hierarchies simultaneously [5]. However, doing automatic merging of operators is an open issue. A third solution attempts to flush the current goal hierarchy whenever a new one needs to be established, while depending on learning to compile goal hierarchies into single operators, and thus to reduce (or eliminate) the overhead of constantly flushing and rebuilding large goal hierarchies [19].

2.3. Support for Learning

The top-down constraint on learning is that it is an important capability for IPs, though not a critical one as yet; while the bottom-up constraint is that learning must occur via chunking, which is Soar's only learning mechanism. In general, Soar systems tightly couple chunking with performance, and the benefits from this tight-coupling have already been illustrated in a variety of Soar systems. However, TacAir-Soar does not perform extensive planning or search (or other non-execution oriented cognitive activities) which is where learning can provide extensive speedups — as explained later, TacAir-Soar is for the most part a model of expert execution behavior. Nonetheless, chunking can be used within the current system to "collapse" the goal hierarchy and compile the knowledge even further for a linear speedup.

Experiments with chunking in TacAir-Soar have however uncovered three interesting issues. The first issue relates to the fact that external actions in this domain take relatively extensive amounts of time compared to the time it takes to expand the goal hierarchy. For instance, consider an IP controlling the helicopter by expanding the goal hierarchy to unmask from a position behind a hill (see Figure 1). Expanding this goal hierarchy takes 0.1 to 0.2 seconds. Chunking could cut down this time to say 0.01 seconds, by providing commands to unmask without requiring the expansion of the hierarchy. However, the execution of the commands in the simulated world takes about 100 seconds. Chunking simply cannot improve upon this execution time — given the speed of the helicopter, it always takes about 100 seconds to unmask. Thus, if we compute the overall speedup in performance due to chunking for the task of unmasking, as is normally done in other Soar systems, it is negligible: 100.1/100.01. Second, typically, in Soar systems, chunks obviate subgoaling since they instantaneously produce results that originally required the subgoaling. However, in TacAir-Soar, the long execution times inhibit the instantaneous reproduction of results. For instance, even after a chunk issues a command to unmask, the helicopter takes 100 seconds to unmask. This delay causes Soar to expand the goal/subgoal hierarchy. Thus, even though chunking seeks to eliminate the expansion of the goal hierarchy, the long execution times prevent immediate feedback and cause the hierarchy to be expanded nonetheless. While neither of the two issues — the negligible speedup, or the expansion of the hierarchy after chunking — illustrate necessarily inappropriate behavior, these are contrary to the effects of chunking on most other Soar systems.

A third issue that comes up in our experiments is the interaction between chunking and the performance of multiple high-level tasks (discussed in the previous section). In particular, chunking is based on the presence of the goal-subgoal hierarchy. Modifications to this hierarchy — either in the form of insertion of independent goals as subgoals of existing goals, or in the form of creation of a forest of goal hierarchies — both interact with chunking. There are many such issues, and addressing them is one of the major outstanding challenges.

There are again two approaches being investigated to address this challenge. The first attempts to modify chunking and its dependence on the single goal hierarchy. The second attempts to maintain chunking in its present form, by suggesting changes in other aspects of the architecture, such as subgoaling. There is much on-going research on this topic [20]. Since learning has not been a critical requirement so far — particularly since expert-level performance is already being modeled — the key design choice here has been to not introduce learning, at least as yet, in TacAir-Soar.

2.4. Psychology: Mimicking Human Capabilities in Design

Exhibiting human-like capabilities is key to the success of IPs in this environment. This includes human-like flexibility in behavior, as well as conformance to human reaction times and human limitations (such as inability to make very hard turns). In particular, the motivation in creating IPs is not to win simulated combat by any and all means available. Rather, it is aid in creating an effective battlefield simulations that provides realistic settings for training humans, developing tactics or evaluating products. For instance, if the IPs react too quickly (or not quickly enough), trainees may learn to act in an excessively defensive (or aggressive) manner. Training in these situations could actually be harmful. Additionally, IPs need to be able to explain their behavior to human experts so that they can gain confidence in IP's capabilities. If experts realize that IPs' are not performing their tasks as humans would, they may not have confidence in IPs' behavior.

Thus, closely mimicking human capabilities in designing IPs is not a choice in this environment, but a necessity. The Soar architecture provides us great support in this endeavor — it is a developing unified theory of cognition, and provides a number of constraints, e.g., on knowledge representation, reaction times, accuracy and learning, that start us off in the right ball park with respect to psychological verisimilitude.

2.5. Real-time Performance

The top-down constraints on real-time performance are: (i) for human-like behavior, IPs must remain within plausible cognitive limits [15] (decisions must occur in less than 300 milliseconds); (ii) for reasonable performance, IPs must remain within simulator time limits (decisions must occur within 500 milliseconds). While these are not hard deadlines, missing them continuously can be fatal. In addition to these constraints on individual decisions, there is also a top-down constraint on the number of decisions before action. In particular, if IPs engage in extensive meta-level reasoning or planning before acting, that can be problematic in time-critical segments of combat. The bottom-up constraints for real-time performance are that at a lower implementation level, all of the knowledge in Soar is represented in the form of rules. All of these rules, plus all of the changes that they cause, must be processed at every decision. Fortunately, the Soar architecture is already heavily optimized with respect to this processing, with a new C-based implementation [9] that is a factor of 15-20 faster than the previous Lisp-based implementation. Nonetheless, rule processing can be a significant burden for real-time performance.

Given these constraints, there were two key design decisions for TacAir-Soar. The first involved attacking the problem of IPs engaging in potentially large numbers of decisions before action. The design decision here was to focus on expert-level performance. This expert-level knowledge allows operators and actions to be selected without impasses occurring in Soar, and therefore without extensive numbers of decisions.

The second design decision here is with respect to the time for individual decisions in TacAir-Soar. We have attempted to minimize the time it takes to process rules by using rule coding styles that help ensure inexpensive rule match [21], and detecting and eliminating unnecessary rule firings. In addition, we have attempted to avoid flooding an IP with unnecessary information, and thus avoid forcing it to perform a lot of low-level computation. For instance, consider an IP controlling a helicopter that is given the task of flying "NAP-of-the-EARTH" (NOE). This involves flying very close to the terrain, about 25 feet above ground level. Here, the IP needs some information about the terrain, so that it does not crash into the (simulated) forests, hills, or other terrain features. Yet, providing the IP a description of every terrain feature in its field of view would be problematic. The solution we have adopted is to provide the IP information about the altitude and location of the highest point within a hundred meters of its direction of flight — the IP then attempts to fly 25 feet above this highest point. Although this topic remains under investigation, preliminary results indicate that the information about this single point is sufficient to enable an IP to fly NOE. This approach may seem inappropriate. However, it turns out that designers of military aircraft use similar strategies to reduce a pilot's cognitive load as much as possible by off-loading many of the low-level, bottom-up computations onto the plane's instruments.

3. Conclusion

This paper described the lessons learned in designing some specific agents — IPs for simulated battlefield environments. For these IPs, the simulated environment is their target environment; we do not intend for them to fly real aircraft! Nonetheless, the IPs face the hard research challenge of integrating a broad range of capabilities (see Figure 2) so that they can participate in simulated combat, both with and against humans. To address this challenge, we have been building a system called TacAir-Soar that is based on the Soar integrated architecture. TacAir-Soar IPs may be considered as having their own architecture, which although distinct from Soar, is nonetheless based on Soar, and constrained by both Soar and the combat simulation environment. The paper discussed the design decisions in TacAir-Soar IPs, and illustrated that: (i) the Soar architecture is capable of providing specific integrated solutions so that TacAir-Soar can easily exhibit a number of the required capabilities; (ii) there are other capabilities for which the Soar architecture provides only a loose framework for their development, or indeed proves somewhat of a hindrance to their development — these have presented an interesting set of research issues, and they may likely contribute to the further evolution of the Soar architecture.

Acknowledgment

We gratefully acknowledge the contribution of the other members of the TacAir-Soar team involved in the creation of the TacAir-Soar system, particularly, Randolph Jones, John Laird, Frank Koss, Lewis Johnson, 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 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 critical support of Cmdr. Dennis McBride of ARPA/ASTO; and the technical support by 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.

Appendix I. Soar

The Soar integrated architecture forms the basis of our work on intelligent automated pilots. In the following, Soar is characterized as a two-level hierarchy of descriptions. The higher, *problem space level* is discussed briefly, followed by a more detailed *architecture level* or *symbol level* description.

At the higher, problem space level of description, Soar can be described as a set of interacting problem spaces. All tasks are formulated as achieving goals in these problem spaces. A problem space consists of states and operators. Within a problem space, knowledge is used to repeatedly select and apply operators to a state to reach a goal. A state in a problem space is a Soar agent's current situation, possibly including perceptions of the external environment as well as internally generated structures. An operator may directly modify this state or it may instigate an external action/event, such as causing an aircraft to turn. The perception of the effects of this external action can then indirectly modify the state. The operator application may be explicitly terminated either upon its successful completion (e.g., the aircraft has turned to the desired heading), or upon a need to be abandon it (e.g., it is necessary to dive instead of continuing to turn the aircraft).

At the lower, architectural (implementation) level of description, Soar's problem space operations can be described in more detail as follows. The knowledge for selection, application and termination of operators is stored in the form of productions (condition-action rules) in Soar's knowledge-base, a production system. Changes in Soar's state, including changes caused by perceptions, can trigger these productions to fire. Upon firing, the productions generate preferences for objects in the state as well as for operators. A preference is an absolute or relative statement of worth of an entity, such as an operator, given the current situation. For instance, a preference might be that an operator to turn left is better than an operator to turn right, given the current situation. Following their generation, Soar's *decision* mechanism examines the operator preferences and selects an operator as the current one. This selection can trigger productions for operator application, which as mentioned above, can modify the state directly or indirectly (by instigating external actions). These modifications may in turn trigger additional productions that further modify the state or generate preferences for the termination of the selected operator and the selection of the next operator. Based on the new set of preferences, the current operator may then be replaced by a different operator. The cycle of operator selection, application and termination can thus continue. However, in some cases, productions generating appropriate preferences may fail to be triggered, e.g., no preferences may be generated for the termination of the selected operator, or for the selection of the next operator. In such cases, an *impasse* occurs.

Soar responds to an impasse by creating a subgoal to resolve it. A new problem space is installed in service of this subgoal. The process of selection and application of operators recurs in this subgoal. If a further impasse occurs while resolving one impasse, a new subgoal is created. With this recursive subgoaling, a whole goal/problem-space hierarchy may be generated. An impasse is resolved if the processing in a subgoal can return the requisite result (leading to a decision in a supergoal) to the original space.

Subgoals and their results form the basis of Soar's sole learning mechanism, *chunking*. Chunking acquires new productions, called *chunks*, that summarize the processing that leads to results of subgoals. A chunk's actions are based on the results of the subgoals. Its conditions are based on those aspects of the goals above the subgoal that are relevant to the determination of the results. Once a chunk is learned, it fires in relevantly similar future situations, directly producing the required result, possibly avoiding the impasse that led to its formation. This chunking process is a form of explanation-based learning (EBL) [13, 16].

References

1. Agre, P. E., and Chapman, D. Pengi: An implementation of a theory of activity. Proceedings of the National Conference on Artificial Intelligence, August, 1987.

2. Bates, J., Loyall, A. B., and Reilly, W. S. Integrating reactivity, goals and emotions in a broad agent. Tech. Rept. CMU-CS-92-142, School of Computer Science, Carnegie Mellon University, May, 1992.

3. Bratman, M. E., Israel, D. J., and Pollack, M. E. "Plans and resource-bounded practical reasoning". *Computational Intelligence 4*, 4 (1988), 349-355.

4. Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., Ceranowicz, A. Z. ModSAF behavior simulation and control. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation, March, 1993.

5. Covrigaru, A. *Emergence of meta-level control in multi-tasking autonomous agents.* Ph.D. Th., University of Michigan, 1992.

6. 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. Collected Papers of the Soar/IFOR project, Spring 1994. Tech. Rept. CSE-TR-207-94, University of Michigan, Department of Electrical Engineering and Computer Science, 1994. Also available as Technical Reports ISI/SR-94-379 from the University of Southern California Information Sciences Institute and CMU-CS-94-134 from Carnegie Mellon University, WWW access http://ai.eecs.umich.edu/ifor/papers/index.html.

7. Jones, R. M., Tambe, M., Laird, J. E., and Rosenbloom, P. Intelligent automated agents for flight training simulators. Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, March, 1993.

8. Jones, R., Laird, J. E., Tambe, M., and Rosenbloom, P. S. Generating behavior in response to interacting goals. Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, May, 1994.

9. Laird, J. E., Congdon, C. B., Altmann, E., & Doorenbos, R. *Soar User's Manual: Version 6.* 1 edition, 1993.

10. Laird, J. E., Newell, A. and Rosenbloom, P. S. "Soar: An architecture for general intelligence". *Artificial Intelligence 33*, 1 (1987), 1-64.

11. The DIS steering committee. The DIS vision: A map to the future of distributed simulation. Tech. Rept. IST-SP-94-01, Institute for simulation and training, University of Central Florida, May, 1994.

12. Maes, P., Darrell, T., Blumberg, B., and Pentland, S. Interacting with Animated Autonomous Agents. Proceedings of the AAAI Spring Symposium on Believable Agents, 1994.

13. Mitchell, T. M., Keller R. M., and Kedar-Cabelli, S. T. "Explanation-based generalization: A unifying view". *Machine Learning 1*, 1 (1986), 47-80.

14. Moravec, H. *Mind Children.* Harvard University Press, Cambridge, Massachusetts, 1990.

15. Newell, A. *Unified Theories of Cognition.* Harvard University Press, Cambridge, Massachusetts, 1990.

16. Rosenbloom, P. S. and Laird, J. E. Mapping explanation-based generalization onto Soar. Proceedings of the Fifth National Conference on Artificial Intelligence, 1986, pp. 561-567.

17. Rosenbloom, P. S. and Laird, J. E. On *Unified Theories of Cognition*: A response to the reviews. In *Contemplating Minds: A Forum for Artificial Intelligence*, W. J. Clancey, S. W. Smoliar, & M. J. Stefik, Eds., MIT Press, Cambridge, MA, 1994, pp. 141-165. (Reprint of Rosenbloom & Laird, 1993, in *Artificial Intelligence*, vol. 59, pp. 389-413).

18. Rosenbloom, P. S., Laird, J. E., Newell, A., and McCarl, R. "A preliminary analysis of the Soar architecture as a basis for general intelligence". *Artificial Intelligence 47*, 1-3 (1991), 289-325.

19. Rubinoff, R., and Lehman, J. Natural language processing in an IFOR pilot. Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, 1994.

20. Laird, J. E. Discussion of external interaction. Panel Discussion at Soar Workshop XIII.

21. Tambe, M. and Rosenbloom, P. Eliminating expensive chunks by restricting expressiveness. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1989, pp. 731-737.

22. Tambe, M., Johnson, W. L., Jones, R., Koss, F., Laird, J. E., Rosenbloom, P. S., and Scwhamb, K. "Intelligent agents for interactive simulation environments". *AI Magazine (To appear)* (1995).

23. Thorpe, J. A., Shiflett, J. E., Bloedorn, G. W., Hayes, M. F., Miller, D. C. The SIMNET network and protocols. Tech. Rept. 7102, BBN systems and technologies corporation, July, 1989.

24. Webber, B., and Badler, N. Virtual interactive collaborators for simulation and training. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation, May, 1993.

Table of Contents

List of Figures

List of Tables