RESC: An Approach for Real-time, Dynamic Agent Tracking

Milind Tambe and Paul S. Rosenbloom
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
Email: {tambe, rosenbloom}@isi.edu
WWW:http://www.isi.edu/soar/{tambe,rosenbloom}

Abstract

Agent tracking involves monitoring the observable actions of other agents as well as inferring their unobserved actions, plans, goals and behaviors. In a dynamic, real-time environment, an intelligent agent faces the challenge of tracking other agents' flexible mix of goal-driven and reactive behaviors, and doing so in real-time, despite ambiguities. This paper presents RESC (REal-time Situated Commitments), an approach that enables an intelligent agent to meet this challenge. RESC's situatedness derives from its constant uninterrupted attention to the *current* world situation — it always tracks other agents' on-going actions in the context of this situation. Despite ambiguities, RESC quickly commits to a single interpretation of the on-going actions (without an extensive examination of the alternatives), and uses that in service of interpretation of future actions. However, should its commitments lead to inconsistencies in tracking, it uses *single-state backtracking* to undo some of the commitments and repair the inconsistencies. Together, RESC's situatedness, immediate commitment, and single-state backtracking conspire in providing RESC its real-time character.
RESC is implemented in the context of intelligent pilot agents participating in a real-world synthetic air-combat environment. Experimental results illustrating RESC's effectiveness are presented.[1]

1 Introduction

In a multi-agent environment, an automated agent often needs to interact intelligently with other agents to achieve its goals. *Agent tracking* — monitoring other agents' observable actions and inferring their unobserved actions, plans, goals and behaviors — is a key capability required to support such interaction.

This paper focuses on agent tracking in real-time, dynamic environments. Our approach is to first build agents that are (reasonably) successful in agent tracking in such environments, and then attempt to understand the underlying principles. Thus, the approach is one of first building an "interesting" system for a complex environment, and then understanding why it does or does not work (see [Hanks *et* a/., 1993] for a related discussion). In step with this approach, we are investigating agent tracking in the context of our on-going effort to build intelligent pilot agents for a real-world synthetic air-combat environment[Tambe *et* a/., 1995]. This environment is based on a commercially developed simulator called ModSAF[Calder *et ai,* 1993], which has already been used in an operational military exercise involving expert human pilots. For an illustrative example of agent tracking in this environment, consider the scenario in Figure 1. It involves two combating pilot agents — L in the light-shaded aircraft and D in the dark-shaded one.

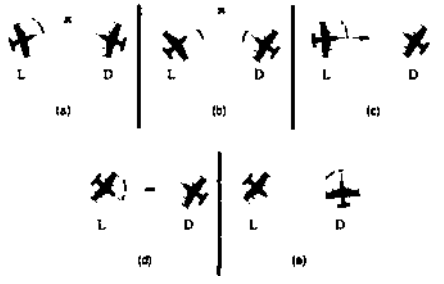

Figure 1: A simulated air-combat scenario. An arc on an aircraft's nose shows its turn direction.

Initially, L and D's aircraft are 50 miles apart, so they can only see each other's actions on radar. For effective performance, they have to continually track these actions. Indeed, D is able to survive a missile attack by L in this scenario due to such tracking, despite the missile being invisible to D's radar. In particular, in Figure 1-a, D observes L turning its aircraft to a collision-course heading (i.e., at this heading, L will collide with D at

[1] We thank Rick Lewis and Yasuo Kuniyoshi for helpful feedback. This research was supported under subcontract to the University of Southern California Information Sciences Institute from the University of Michigan, as part of 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).

the point shown by x). Since this heading is often used to reach one's missile firing range, D infers the possibility that L is trying to reach this range to fire a missile. In Figure 1-b, D turns its aircraft 15° right. L reacts by turning 15° left, to maintain collision course. In Figure 1-c, L reaches its missile range, points its aircraft at D's aircraft and fires a radar-guided missile. While D cannot see the missile on its radar, it observes L's turn, and infers it to be part of L's missile firing behavior. Subsequently, D observes L executing a 35° turn away from its aircraft (Figure 1-d). D infers this to be an *fpole* turn, typically executed after firing a missile to provide radar guidance to the missile, while slowing the closure between the two aircraft. While D still cannot observe the missile, it is now sufficiently convinced to attempt to evade the missile by turning 90° relative to the direction of L's aircraft (Figure 1-e). This *beam* turn causes D's aircraft to become invisible to L's (doppler) radar. Deprived of radar guidance, L's missile is rendered harmless.

Meanwhile, L tracks D's beam turn in Figure 1-e, and prepares counter-measures in anticipation of the likely loss of both its missile and radar contact.

Thus, the pilot agents need to continually track their opponents' actions, such as turns, and infer unobserved actions, high-level goals and behaviors, such as the fpole, beam or missile firing behaviors. This agent tracking capability is related to plan-recognition[Kautz and Allen, 1986; Azarewicz *et a*/., 1986]. The key difference is that plan-recognition efforts typically focus on tracking a narrower (plan-based) class of agent behaviors, as seen in static, single-agent domains. In particular, they assume that agents rigidly follow plans step-by-step. In contrast, agent tracking involves the novel challenge of tracking a broader mix of goal-driven and reactive behaviors. This capability is important for dynamic environments such as air-combat simulation where agents do not rigidly follow plans — as just seen, pilot agents continually react to each other's maneuvers.

Agent tracking and plan recognition are both part of a larger family of comprehension capabilities that enable an agent to parse a continuous stream of input from its environment, whether it be in the form of natural language or speech or music or simulated radar input, as is the case here (e.g., see [Rich and Knight, 1990, chapter 14]). Resolving ambiguities in the input stream is a key problem when parsing all of these different types of input. One example of the ambiguity faced in agent tracking can be seen in L's turn in Figure 1-c. From D's perspective, L could be turning to fire a missile. Alternatively, L could be beginning a 180° turn to run away from combat. Or L could simply be following its flight plan, particularly if it has a much shorter radar range, and thus is likely unaware of D. Despite such ambiguities, D has to track L's actions with sufficient accuracy so as to respond appropriately. The novel challenge in this domain — at least with respect to previous work in plan recognition — is that the ambiguity resolution has to occur in real-time. As the world rapidly moves on, an agent cannot lag behind in tracking. Thus, if D is late or inaccurate in its tracking of L's missile firing maneuvers in Figure 1-c, it may not evade the missile in time.

This paper describes an approach called RESC (REal-time Situated Commitments) for agent tracking that addresses the above challenges. RESC's situatedness rests on its constant attention to the current world situation, and its tracking of other agents' actions in the context of this situation. Despite its situatedness, RESC does make some commitments about the other agent's unobservable actions, behaviors and goals, and attempts to use those in tracking the agent's future actions. In ambiguous situations, these commitments could be inappropriate and could lead to failures in tracking — in such cases, RESC modifies them on-line, without re-examining past world states. Together, RESC's situatedness, immediate commitments (despite the ambiguities), and its on-line modification of commitments provide RESC its real-time character.

In the following, we first describe the process that RESC employs for tracking other agent's flexible and reactive behaviors (Section 2). This process enables RESC to be situated in its present as it tracks an agent's actions. Subsequently, RESC's ambiguity resolution and real-time properties are described in Section 3. These descriptions are provided in concrete terms, using an implementation of the pilot agents in a system called TacAir-Soar[Tambe *et a*/., 1995J, built using the Soar architecture[Newell, 1990; Rosenbloom *et ai,* 1991]. We assume some familiarity with Soar's problem-solving model, which involves applying operators to states to reach a desired state.

2 Tracking Flexible Goal-driven and Reactive Behaviors

In an environment such as air-combat simulation, agents possess similar behavioral flexibility and reactivity. Thus, the (architectural) mechanisms that an agent employs in generating its own behaviors may be used for tracking others' flexible and reactive behaviors. Consider, for instance, D's tracking of L's behaviors in Figure 1-c. D generates its own behavior using the operator hierarchy shown in Figure 2-a. (The solid lines indicate the actual hierarchy, and the dashed lines indicate unselected options.) Here, at the top-level, D is executing its mission — to protect its home-base for a given time period — via the *execute-mission* operator. Since the termination condition of this operator — completion of D's mission — is not yet achieved, a subgoal is generated.[2] D rejects options such as*fjollow-flight-plan* and *run-away* in this subgoal in favor of the *intercept* operator, so as to combat L. In service of *intercept,* D selects *employ-missile* in the next subgoal. However, since D has not reached its missile firing range and position, it selects *get-firing-position* in the next subgoal. Skipping to the final subgoal, *maintam-heading* enables D to maintain

[2] A Soar operator has termination conditions — if the operator's application (or new sensor input) changes the state so as to satisfy the termination conditions, then that operator and all of its subgoals are terminated. If the termination conditions remain unsatisfied, a subgoal is created, within which new operators are applied.

its heading, as seen in Figure 1-c.

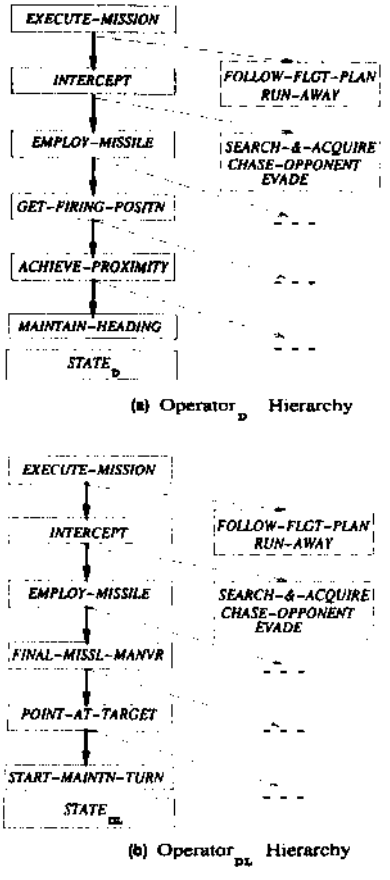

Figure 2: Operator hierarchies: Solid lines indicate actual selections; dashed indicate unselected options.

The operators used for generating D's own actions, such as in Figure 2-a, will be denoted with the subscript D, e.g., $intercept_D$. $Operator_D$ will denote an arbitrary operator of D. $State_D$ will denote the global state shared by all of these operators. It maintains all of the dynamic sensor input regarding D's own aircraft, such as its heading and altitude. It also maintains dynamic radar input regarding L's aircraft, such as heading, range, collision course and other geometric relationships. Additionally, it maintains non-sensor information, e.g., D's missile capabilities. Together, $state_D$ and the $operator_D$ hierarchy constitute the introspectable aspect of D, and in this sense may be considered as D's model of its present self, referred to as $model_D$.

$Model_D$ supports D's flexible/reactive behaviors via its embedding within Soar; and in particular, via two of Soar's architectural features: (i) a decision procedure that supports flexibility by integrating all available knowledge about absolute or relative worth of candidate operators right before deciding to commit to a single operator; (ii) termination conditions for operators that support reactivity by terminating operators in response to the given situation[Rosenbloom *et al.*, 1991]. The point here is not that these specific architectural features are the only way to yield such behavior, but rather that there are such features, and that they can be reused in tracking other agents' behaviors. To illustrate this re-use, we assume for now that D and L possess an identical set of maneuvers. (Note that this sameness of maneuvers is not necessary; all that is required is for D to have an accurate model of its opponent's maneuvers.)

Thus, D uses a hierarchy such as the one in Figure 2-b to track L's behaviors. Here, the hierarchy (the solid lines in Figure 2-b) represents D's model of L's current operators in the situation of Figure 1-c. These operators are denoted with the subscript DL. This $operator_{DL}$ hierarchy, and the $state_{DL}$ that goes with it, constitute D's model of L or $model_{DL}$. Within $model_{DL}$, *execute-mission-Qij* denotes the operator that D uses to track L's mission execution. Since L's mission is not yet complete, D applies the $intercept_{DL}$ operator in the subgoal to track L's intercept. The unselected alternatives here, e.g., $run\text{-}away_{DL}$ indicate the ambiguity in tracking L's actions (however, assume for now that this is accurately resolved). In the next subgoal, $employ\text{-}missile_{DL}$ is applied. Since L has reached its missile firing position, in the next two subgoals, $final\text{-}missile\text{-}maneuver_{DL}$ tracks L's final missile maneuver, and $point\text{-}at\text{-}target_{DL}$ tracks L's turning to point at D. In the final subgoal, D applies the $stari\text{-}\&\text{-}maintatn\text{-}turn_{DL}$ operator to $state_{DL}$, which does not (can not) actually cause L turn. Instead, this operator predicts L's action and matches the prediction against L's actual action. Thus, if L starts turning to point at D's aircraft, then there is a match with $model_{DL}$'s predictions — D believes L is turning to point at its target, D, to fire a missile. When L's aircraft turns sufficiently to point straight at D's aircraft (Figure 1-c), the termination condition of the $point\text{-}at\text{-}target_{DL}$ operator is satisfied, and it is terminated. A new operator, $push\text{-}fire\text{-}button_{DL}$, is then applied in the subgoal of $final\text{-}misstle\text{-}maneuve_{DL}$. This operator predicts a missile firing, although the missile cannot actually be observed. $State_{DL}$ maintains a representation of the missile, and marks it with a low likelihood. Following that, the $fpole\text{-}right_{DL}$ operator predicts L's right turn for an fpole. When this prediction is matched with L's turn in Figure 1-d, the missile's likelihood is changed to high. D now attempts to evade the missile, with $beam\text{-}right_D$. (D currently chooses arbitrarily between the execution of $operator_D$ and $operator_{DL}$, as it generates its own actions, while also tracking L s actions.)

The above agent tracking process is related to previous work on *model tracing* in intelligent tutoring systems(ITS) for tracking student actions[Anderson *et al.*, 1990; Ward, 1991]. However, that work has primarily focused on static environments. A recently developed ITS, REACT[Hill and Johnson, 1994], extends model tracing to a more dynamic environment. REACT relies upon a plan-driven tracking strategy, and deals with the more dynamic aspects of the domain as special cases. It specifically abstracts away from tracking students' mental states. In contrast, pilots appear to track their op-

ponents' behaviors in more detail. Such tracking is supported here via a uniform apparatus for the generation of an agent's own flexible/reactive behaviors and tracking other agents' behaviors. In particular, $operator_{D}$ and $operator_{DL}$ are selected and terminated in the same flexible manner. Thus, as $state_{DL}$ changes, which it does in reflecting the changing world situation, new $operators_{DL}$ may get selected in response. This is key to RESC's situatedness — L's on-going actions are continually tracked in the context of the current $state_{DL}$. For further details on this tracking technique, please see[Tambe and Rosenbloom, 1995].

3 Real-time Ambiguity Resolution

Ambiguity manifests itself in two forms in the agent tracking process introduced in the previous section. One form involves the alternative operators available for tracking the other agent's actions, as seen in the dashed boxes in Figure 2-b. Given these alternatives, it is difficult to make accurate selections of $operator_{DL}$, such that their predictions successfully match L's actions. Should an $operator_{DL}$ selection be inaccurate, in typically results in a *match failure* (if not immediately, then in some further $operator_{DL}$ application). Thus, in Figure 2-b, the $operator_{DL}$ hierarchy predicts L will turn to point at D's aircraft. Suppose this prediction is inaccurate, and L turns in the opposite direction. This difference in the anticipated and actual action leads to a match failure, indicating an inaccuracy in tracking. Similar match failures can also occur if L fails to begin (or stop) turning or maintain heading as anticipated.

A second form of ambiguity is seen in $state_{DL}$. $State_{DL}$ needs to maintain the same types of information as are in $state_{D}$. Here, there is ambiguity related to both the dynamic sensor information and the static non-sensor information. With respect to static information, there are ambiguities about L's radar and missile capabilities. Even if these are resolved, there are ambiguities about dynamic information, such as whether L has detected D on radar. For instance, in Figure 1-a, based on the static radar range information, D assumes it has arrived within L's radar range; but L may or may not have detected D, depending on its radar's orientation. Such ambiguities in $state_{DL}$ are intimately connected to ambiguities in $operator_{DL}$, since the $operator_{DL}$ hierarchy is dependent on the current $state_{DL}$. Thus, if D assumes it is not detected on L's radar, then the $intercept_{DL}$ operator is ruled out, since there is nothing for L to intercept. In contrast, if D assumes L has detected it, then $intercept_{DL}$ is a likely possibility. A subgoal of $intercept_{DL}$ predicts L's turn to collision course, which is matched by L's turn in Figure 1-a — D now believes L has detected it, and L is going to collision course to intercept. Note that, if D believes that L has detected it, $state_{DL}$ needs to maintain the various dynamic inputs that D believes L obtains from its radar regarding D's heading, range, geometric relationships etc. Fortunately, many of these quantities are symmetric and can be reused from corresponding quantities in $state_{D}$.

It is difficult to resolve the above ambiguities using methods that have been previously suggested in the model tracing literature. Ward[Ward, 1991] notes that previous model tracing systems have mostly relied on communication with the modeled agent to resolve ambiguities. In air-combat simulation, such communication with enemy pilots is clearly ruled out. Ward's solution in the absence of such information is an exhaustive backtrack search of all of the different alternatives. In the example in 2-b, this involves an attempt to execute and match other operator hierarchies — generated by alternatives such as $run\text{-}away_{DL}$ or $follow\text{-}flight\text{-}plan_{DL}$ — via a systematic backtrack search before committing to $intercept_{DL}$. Unfortunately, this search-then-commit approach will very likely cause tracking to lag far behind the rapidly changing world, precluding D from responding to L's maneuvers in real-time. Furthermore, given the volume of dynamic information on $state_{DL}$, the maintenance of multiple old copies of $state_{DL}$ for backtracking could itself consume non-trivial amounts of space and time. Parallel real-time search of alternative $models_{DL}$ could eliminate the backtracking; however, we will focus on a sequential solution given the implementation technology available to us. Furthermore, parallelism may not be adequate when faced with the expected combinatorics in the number of alternatives. Borrowing ambiguity resolution methods from the plan recognition literature would be yet another possibility; but the computational costs (intractability) of techniques such as automated deduction[Kautz and Allen, 1986] are a significant concern.

So instead, we propose a new approach called RESC (REal-time Situated Commitments) that addresses the above concerns. As seen earlier, RESC's situatedness arises from its tracking of L's on-going actions and behaviors in the context of the current $state_{DL}$. RESC's commitment is to a single $model_{DL}$, with a single $state_{DL}$ that records the on-going world situation in real-time and a single $operator_{DL}$ hierarchy, that provides an on-going interpretation of an opponent's actions. Given the intense real-time pressure, RESC does not spend time trying to match alternatives; instead, it just commits to a single $operator_{DL}$ hierarchy, and any facts inferred in $state_{DL}$ due to this hierarchy. It then tries to use these commitments as context for tracking L's future actions. However, in some cases, the commitments may get withdrawn given RESC's situatedness — as $state_{DL}$ changes, it may satisfy the termination conditions of an $operator_{DL}$ and thus cause it, and all of its subgoals, to terminate.

When faced with ambiguity, it is possible that RESC commits to an inaccurate $operator_{DL}$ and $state_{DL}$, leading to a match failure. RESC recovers from such failures by relying on a method called *single-state backtracking*, that undoes some of its commitments, resulting in the generation of new $operator_{DL}$ hierarchies, in real-time. Of course, if RESC makes more intelligent commitments in the first place — by reducing the ambiguity in the situation with which it is faced — there will be less of a need for undoing its commitments. Subsection 3.1 first describes strategies — some general and some domain specific — used for reducing ambiguities in both $state_{DL}$ and $operator_{DL}$. Subsection 3.2 then

describes single-state backtracking.

3.1 Reducing Ambiguities

There are two classes of strategies used in RESC to resolve ambiguities: *active* and *passive*. The active strategies rely upon an agent's active participation in its environment to gather information to resolve ambiguities. In particular, an automated pilot, such as D, can act in its environment and force its opponent L to react and provide disambiguating information. Consider again the example in Figure 1-a. As discussed earlier, D faces ambiguity in siateDL about whether L's radar has detected D. This gets resolved with L's turn to collision course. Unfortunately, if L just happens to be on collision course, it may not turn any further, and the ambiguity would be more difficult to resolve. In such cases, D can randomly turn 15-20°, as shown in Figure 1-b, causing L to react if it wishes to maintain collision course. This provides D sufficient disambiguating information — L's radar has detected D. Unfortunately, D's actions in service of active ambiguity resolution may interfere with its other goals, such as firing a missile at L. In general, such interference is difficult to resolve. Therefore, currently, active ambiguity resolution is based on a fixed set of known maneuvers (supplied by human experts).

In contrast, passive ambiguity resolution strategies rely on existing information to resolve ambiguities. One key piece of information is that in this hostile environment, an opponent is likely to engage in the most harmful maneuver. This information is used in the form of a *worst case* strategy for disambiguation. Thus, given a choice, D always selects the worst-case operatorDL (from its own perspective) while tracking L's actions. For instance, if there is ambiguity between *run-awayDL* or *interceptDL ,* D will select *intercept DL,* which is more harmful. Similarly, D resolves ambiguity in the static information in stateDL via the worst-case strategy, e.g., it assumes that L's aircraft is carrying the most powerful missiles and radar that it can carry. Unfortunately, this worst-case strategy can lead to overly pessimistic behavior. In the absolute worst-case, the only option for D is to run away. Therefore, D applies it selectively, typically in cases where it has to disambiguate rapidly, and yet no other means are available. Thus, as seen above, D does not automatically assume detection by L's radar, even though that would be the worst-case assumption.

A second passive ambiguity resolution strategy is *test incorporation[Bennett* and Dietterich, 1986]. The key idea is to generate fewer incorrect alternatives in ambiguous situations. In particular, modelDL generates alternative operatorsDL that are tested by matching against L's actual actions. Observations regarding these actions can be used to avoid generating alternatives that are guaranteed to lead to match failures. For instance, in Figure 1-d, *fpole-rightDL* and fpole-*left-fii,* are two alternatives available to D in tracking L's actions. If D already sees L turning to its right, then *fpole-leftDL* can be eliminated, since it would be guaranteed to lead to a match failure. Test incorporation relies on such spatial relationships.

A third passive ambiguity resolution strategy is *goal incorporation* (e.g., see [Van Beek and Cohen, 1991]). The key idea here is to resolve ambiguities only to the extent necessitated by an agent's goals. For example, given the reality of the simulation environment, L's aircraft often unintentionally deviates from its intended heading. Given such deviations, L sometimes makes corrections to its headings. However, D does not really need to track and disambiguate these small deviations and corrective actions. It therefore uses *fuzz-box* filters that disregard specified deviations in L's actions. For instance, for *point-at-targetDL,* which tracks L's pointing maneuver (Figure 1-c), the fuzz-box filter disregards 5° of deviation in L's heading. Such filtering also helps to avoid tracking of detailed aspects of stateDL, and avoids ambiguities there.

3.2 Single-State Backtracking in RESC

Based on the above disambiguation strategies, RESC commits to a single stateDL and a single operatorDL hierarchy, which track L's actions as described in Section 2. However, should this cause a match failure, single-state backtracking is used to undo some commitments. As its name suggests, this backtracking takes place within the context of a single stateDL . Starting from the bottom of the operatorDL hierarchy, operators are terminated one by one in an attempt to get alternatives to take their place. Some alternatives do get installed in the hierarchy, and possibly change stateDl , but lead to match failures. These are also replaced, until some alternative leads to an operator DL hierarchy that culminates in match success.[3]

Why is this process real-time? The main reason is that backtracking occurs without a re-examination of past sensor input or mental recreation of older statesDL. In particular, while backtrack search would normally involve revisiting old statesDl and reconsidering the different operatorsDL possible in each of those states — creating an opening for combinatorics — RESC completely avoids such computation. Furthermore, although RESC does backtrack over the operator hierarchy, there are three factors that ameliorate the combinatorics there. First, given RESC's situatedness, backtracking remains tied to the present stateDL. Thus, while a match failure is recognized and the oacktrack process begun, L and D's aircraft continue to move and turn, changing their speeds, headings, altitudes, and relative geometric relationships (e.g., range, collision course, etc). StateDL is continuously updated with this latest information. The backtracking process takes place in the context of this continuously changing state. Thus, only those alternative operatorsDL that are relevant to the current stateDL get applied. Similarly, in some cases, changes in stateDL cause portions of the operatorDL hierarchy to terminate automatically during the backtrack process. In other words, RESC is continuously dragged forward as the world changes. Second, RESC does not oblige D to address the match failure before D can execute

[3] In a few cases, there are pending changes related to ambiguities in stateDL, e.g., has L detected D? These are applied first, hoping they cause changes to operatorDL and lead to success.

any of its own operatorsD. Thus D is free to act to the extent it can. Finally, indeed, if the world were to magically become static, RESC's strategy will result in a complex search, although still within the context of a single stateDL . However, it is unclear if this is necessarily problematic — a static world should possibly merit a more thorough search.

Let us consider some examples of single-state backtracking. As a simple example, suppose D has commited to the modelDL in Figure 2-b. Initially, *point-at-target DL* has match success in that, as predicted, L indeed starts turning towards D(see Figure 3-a for an illustration). However, L really has decided to run away; so it continues turning 180° without stopping when pointing at D (Figure 3-b). This leads to a match failure in the operatorDL hierarchy. Single-state backtrackingnow ensues, terminating operators beginning from the bottom of the hierarchy. Finally, *interceptDL,* is terminated and replaced by *run-away DL.* This predicts L to be turning towards its home-base, which successfully matches L's actions (Figure 3-c). Thus, D successfully applies *run-away DL,* predicting and matching L's actions, without mentally recreating the state DL in which L may have initiated its run-away maneuver.

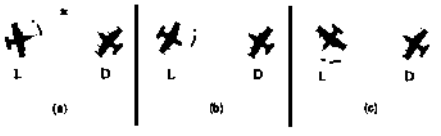

Figure 3: L continues to turn to run away.

A slightly more complex example involves situations where L is engaging in a beam maneuver. Here, D initially matches *fpole-rightDL,* and even infers L's missile firing, as part of stateDL . However, as L keeps turning, there is soon a matcn failure, causing D to backtrack until *beam-right DL* successfully matches. There are two key points here. First, again D is successful in applying *beam-rightDL,* without mentally recreating the stateDL in which L may have initiated its beam maneuver. Second, D's earlier inference of L's missile firing is not removed, even though it is based on a sequence of operators that eventually led to a match failure. This is because it is difficult for D to decide if L was initially maneuvering to fire a missile and then switched to beam, or if it was always engaged in beam. Not knowing any better, D does not eliminate the earlier inference from stateDL. Fortunately, when aircraft turn 90° to beam, they cannot provide radar guidance to their missiles. Therefore, with L's beam, D infers that the missile that it earlier inferred on stateDL has lost guidance and become harmless. The end result is identical to a case where D had successfully tracked L's beam maneuver, without the failed intermediate inference of an *fpole-rightDL* maneuver.

We have so far found RESC's single-state backtracking to be successful in the air-combat simulation domain (see Section 4). Given the potential application of this approach for other areas of real-time comprehension, it is useful to analyze the reasons behind its success. Towards this end, consider first the following simplified and abstract characterization of a successful application of single-state backtracking: **L** initiates some maneuver β at time *T0.* However, at *T0,* **D** attempts to match it by applying an operator $\alpha_{\mathbf{DL}}$ to $\text{state}_{\mathbf{DL}}^{T0}$ (which denotes $\text{state}_{\mathbf{DL}}$ at time *T0*). At time $T0+\tau$, in $\text{state}_{\mathbf{DL}}^{T0+\tau}$ **D** recognizes a match failure with $\alpha_{\mathbf{DL}}$. It backtracks and applies $\beta_{\mathbf{DL}}$ to $\text{state}_{\mathbf{DL}}^{T0+\tau}$. The key observation here is that despite the time delay τ and the intervening application of $\alpha_{\mathbf{DL}}$, $\beta_{\mathbf{DL}}$ is successful in predicting and matching **L**'s maneuvers, as though it were applied to $\text{state}_{\mathbf{DL}}^{T0}$. Based on this observation, in terms of operator preconditions and effects, we can infer at least three requirements for single-state backtracking to work. In the following, we list these requirements, and illustrate how pilot agents currently adhere to them:

1. *The preconditions of* $\beta_{\mathbf{DL}}$ *are satisfied in* $\text{state}_{\mathbf{DL}}^{T0+\tau}$ *in much the same way as in* $\text{state}_{\mathbf{DL}}^{T0}$: For pilot agents, operator preconditions are expressed so they do not test specific positions of aircraft, but rather abstracted features of $\text{state}_{\mathbf{DL}}$ — similar to the fuzz-boxes in Section 3.1 — that are unlikely to change in τ. That is, abstracted features tested by preconditions change at a rate smaller than $1/\tau$.

2. *The effects of* $\beta_{\mathbf{DL}}$ *when applied to* $\text{state}_{\mathbf{DL}}^{T0+\tau}$ *are equivalent to the effects of* $\beta_{\mathbf{DL}}$ *when applied to* $\text{state}_{\mathbf{DL}}^{T0}$: This is achieved by expressing operator effects relative to some feature of $\text{state}_{\mathbf{DL}}$ that is unlikely to have changed in the intervening time period τ. For instance, the effect of $\textit{run-away}_{\mathbf{DL}}$ predicts **L** is headed towards its home base — the location of this base is unlikely to change within τ. Similarly, the effects of operators such as $\textit{beam-right}_{\mathbf{DL}}$ are expressed in relative terms as **L** turning to achieve 90° angle-off, which is an angle formed by **L**'s heading relative to the straight line joining **D** and **L** (while this line does change its position in τ, given the range between **D** and **L** the change is small, and gets absorbed by the fuzz-boxes). If the effects were expressed instead as turning 90° from **L**'s current heading, they would have provided very different results at *T0* and $T0+\tau$, defeating single-state backtracking. Overall, the above seems possible because operators in this environment typically strive to achieve positions relative to slowly changing reference points, such as turning to a particular heading relative to an opponent's aircraft, or relative to a waypoint such as the home-base.

3. *The effects of* $\alpha_{\mathbf{DL}}$ *as applied to* $\text{state}_{\mathbf{DL}}^{T0}$ *are eliminated at some time* $T0+\tau$ *before they cause inconsistencies in* **D***'s response*: As seen in an example above, even though **L**'s missile firing was inferred, and this inference was not "cleaned up" upon recognition of a match failure, **L**'s future maneuver automatically nullifies the effect of that inference. Fortunately, typical $\text{operator}_{\mathbf{DL}}$ applications do not

commit to such inferences on state$_{DL}$. For those that do commit, these commitments get removed by future maneuvers or become irrelevant.

In some cases, L quickly terminates its maneuver β within time period τ, and initiates a new one γ at time $T0+\tau$. Here, given RESC's situatedness, at time $T0+\tau$, D completely skips tracking β_{DL}, and tracks γ_{DL} instead. Fortunately, since D's initial attempt is to apply worst-case operators$_{DL}$, there is at least the assurance that what is skipped (β_{DL}) is not among the worst of the possibilities.

4 Implementation and Evaluation

To understand the effectiveness of the RESC approach, we have implemented it as part of an experimental variant of TacAir-Soar[Tambe *et al.*, 1995]. The current TacAir-Soar system contains about 2000 rules, and automated pilots based on it have participated in combat exercises with expert human pilots. Our experimental version — created to investigate the RESC approach — contains about 950 of these rules. Proven techniques from this experimental version, called TacAir-Soar^{RESC} are transferred back into the original TacAir-Soar.

There are at least two aspects to understanding the effectiveness of TacAir-Soar^{RESC}. The first aspect is whether the current approach enables D, the TacAir-Soar^{RESC} pilot agent, to track its opponents' actions accurately in real-time. We conducted two sets of experiments to address this issue. The first set involved running Soar-vs-Soar air-combat simulation scenarios (as outlined by the human experts). The results from these experiments are presented in Table 1.

Scn. num	Num oppnts	Total operts	% operators agent track	% of colm 2 in match fail
1	1	37	8%	0%
2	1	133	45%	17%
3	2	167	50%	16%
4	2	175	54%	17%
5	4	142	63%	11%

Table 1: Results of Soar-vs-Soar experiments.

The first column lists the scenario number. The second column lists the number of opponents that D faces in each scenario — this varies from one to four in these scenarios. The third column indicates the total number of D's operator executions in each scenario. This includes operators for D's own actions, as well as for tracking opponent actions. The total number provides some indication of the comparative complexity of the different scenarios. Note that these operators are not all applied in a sequence at regular intervals; D often waits in between applications as it tries to get into different positions. Indeed, despite the differences in the number of operators, the total time per scenario is about the same, approx 5 minutes. The fourth column shows the percentage of operator executions involved in agent tracking. This percentage clearly depends on the number of maneuvers that the opponents execute, and the number of opponents. The key point here is that agent tracking is a non-trivial task for D. Furthermore, higher percentages of operator executions may be dedicated to agent tracking with increased numbers of opponents.

The fifth column shows the percentage of agent tracking operators involved in match failures (counting operators at the bottom of the hierarchy that encountered the failure, but not their parents). The main point here is that the overall percentage of these operator is low; at most 17% of the agent tracking operators are involved in match failures.

In all of these cases, D is successful in tracking opponents in real-time so as to react appropriately. Even in cases where D encounters match failures, it is able to backtrack to track the on-going activities in real-time and respond appropriately. However, as the number of opponents increases, D does face resource contention problems. With four opponents, it is unable to track the actions of all of the agents in time, and gets shot down (hence fewer operators). This resource contention issue is under active investigation[Tambe, 1995].

Our second set of experiments involved Soar-vs-ModSAF simulated air-combat scenarios. ModSAF-based[Calder *et al.*, 1993] pilot agents are controlled by finite state machines combined with arbitrary pieces of code, and do not exhibit high behavioral flexibility. While D was in general successful in agent tracking in these experiments —■ it did recognize the maneuvers in real-time and respond to them — one interesting issue did come up. In particular, in one of the scenarios here, there was a substantial mismatch in D's worst assumptions regarding its opponent's missile capabilities and the actual capabilities — leading to tracking failures. Dealing with model mismatch is also an issue for future work.

The second aspect to understanding the effectiveness of TacAir-Soar ^{RESC} is some quantitative estimate of the impact of agent tracking on improving D's overall performance. In general, this is a difficult issue to address (see for instance the debate in [Hanks *et al.*, 1993]). Nonetheless, we can at least list some of the types of benefits that D accrues from this capability. First, agent tracking is crucial for D's survival. Indeed, it is based on agent tracking that D can recognize an opponent's missile firing behavior and evade it. Second, agent tracking improves D's overall understanding of a situation, so it can act/react more intelligently. For instance, if an opponent is understood to be running away, D can chase it down, which would be inappropriate if the opponent is not really running away. Similarly, if D is about to fire a missile, and it recognizes that the opponent is also about to do the same, then it can be more tolerant of small errors in its own missile firing position so that it can fire first. Finally, agent tracking helps D in providing a better explanation of its behaviors to human experts. (Such an explanation capability is currently being developed[johnson, 1994]). If human experts see D as performing its task with an inaccurate understanding of opponents' actions, they will not have sufficient confidence to actually use it in training.

5 Lessons Learned

This paper presented an approach called RESC, for agent tracking in real-time dynamic environments. Our investigation was based on a real-world synthetic environment that has already been used in a large-scale operational military exercise[Tambe *et ai,* 1995]. Lessons learned from this investigation — as embodied in RESC — are as follows:

- *To track other agents' flexible and reactive behaviors:* Reuse the architectural mechanisms that support an agent's own flexible/reactive behaviors in service of tracking others' behaviors.
- *To address ambiguities in real-time:* Quickly commit to a single interpretation, and use single-state backtracking to recover from erroneous commitments.
- *To address real-time issues in general:* Keep tracking firmly tied to the *now,* i.e., to the present state.

One key issue for future work is investigating the generality of these lessons by applying RESC to other competitive and collaborative multi-agent domains. One candidate that has been suggested is a real-time multi-robot domain where robots track other robots or humans to collaborate in a task by observation (rather than by communication)[Kuniyoshi *et ai,* 1994]. Beyond agent tracking, there is some indication that RESC could apply in other real-time comprehension tasks. For instance, a RESC-type strategy has been previously used in a real-time language comprehension system[Lewis, 1993]. This system also commits to a single interpretation of an input sentence despite ambiguity, and attempts to repair the interpretation in real-time when faced with parsing difficulties. We hope that investigating these broader applications will lead to an improved understanding of agent tracking and comprehension.

References

[Anderson *et aL,* 1990] J. R. Anderson, C. F. Boyle, A. T. Corbett, and M. W. Lewis. Cognitive modeling and intelligent tutoring. *Artificial Intelligence,* 42:7-49,1990.

[Azarewicz *et ai,* 1986] J. Azarewicz, G. Fala, R. Fink, and C. Heithecker. Plan recognition for airborne tactical decision making. In *Proceedings of the National Conference on Artificial Intelligence,* pages 805-811. Menlo Park, Calif.: AAAI press, 1986.

[Bennett and Dietterich, 1986] J. S. Bennett and T. G. Dietterich. The test incorporation hypothesis and the weak methods. Technical Report 86-30-4, Department of Computer Science, Oregon State University, 1986.

[Calder *et ai,* 1993] R. B. Calder, J. E. Smith, A. J. Courtemanche, J. M. F. Mar, and A. Z. Ceranowicz. Modsaf behavior simulation and control. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation,* 1993.

[Hanks *ei ai,* 1993] S. Hanks, M. E. Pollack, and P. R. Cohen. Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *AI Magazine,* 14(4):17-42, 1993.

[Hill and Johnson, 1994] R. Hill and W. L. Johnson. Situated plan attribution for intelligent tutoring. In *Proceedings of the National Conference on Artificial Intelligence.* Menlo Park, Calif.: AAAI press, 1994.

[Johnson, 1994] W. L. Johnson. Agents that learn to explain themselves. In *Proceedings of the National Conference on Artificial Intelligence,* Seattle, WA, August 1994. Menlo Park, Calif.: AAAI press.

[Kautz and Allen, 1986] A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the National Conference on Artificial Intelligence,* pages 32-37. Menlo Park, Calif.: AAAI press, 1986.

[Kuniyoshi *et ai,* 1994] Y. Kuniyoshi, S. Rougeaux, M. Ishii, N. Kita, S. Sakane, and M. Kakikura. Cooperation by observation - the framework and the basic task pattern. In *Proceedings of the IEEE International Conference on Robotics and Automation,* May 1994.

[Lewis, 1993] R. L. Lewis. An architecturally-based theory of human sentence comprehension. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* 1993.

[Newell, 1990] A. Newell. *Unified Theories of Cognition.* Harvard Univ. Press, Cambridge, Mass., 1990.

[Rich and Knight, 1990] E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, New York, NY, 1990.

[Rosenbloom *et ai,* 1991] P. S. Rosenbloom, J. E. Laird, A. Newell, , and R. McCarl. A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence,* 47(l-3):289~325, 1991.

[Tambe and Rosenbloom, 1995] M. Tambe and P. S. Rosenbloom. Event tracking in a dynamic multi-agent environment. *Computational Intelligence,* (To appear), 1995. WWW: http://www.isi.edu/soar/tambe/event.html.

[Tambe *et ai,* 1995] M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine,* 16(1), Spring 1995.

[Tambe, 1995] M. Tambe. Recursive agent and agent-group tracking in a real-time dynamic environment. In *Proceedings of the International Conference on Multt-agent systems (ICMAS),* June 1995.

[Van Beek and Cohen, 1991] P. Van Beek and R. Cohen. Resolving plan ambiguity for cooperative response generation. In *Proceedings of International Joint Conference on Artificial Intelligence,* pages 938-944, 1991.

[Ward, 1991] B. Ward. *ET-Soar: Toward an ITS for Theory-Based Representations.* PhD thesis, School of Computer Science, Carnegie Mellon Univ., 1991.