University
of Southern
California

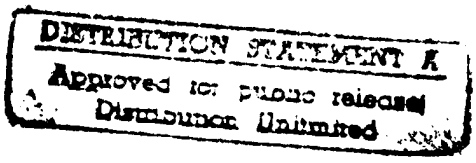
# Event Tracking in a Dynamic Multi-agent Environment

Milind Tambe
Paul S. Rosenbloom

USC/Information Sciences Institute

September 1994

ISI/RR-94-393






INFORMATION
SCIENCES
INSTITUTE

310/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

# Event Tracking in a Dynamic Multi-agent Environment

Milind Tambe
Paul S. Rosenbloom

USC/Information Sciences Institute

September 1994

ISI/RR-94-393

# REPORT DOCUMENTATION PAGE

*FORM APPROVED*
*OMB NO. 0704-0188*



| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | September 1994 | Research Report |

**4. TITLE AND SUBTITLE**

Event Tracking in a Dynamic Multi-agent Environment

**5. FUNDING NUMBERS**

N00014-92-K-2015

**6. AUTHOR(S)**

Milind Tambe and Paul S. Rosenbloom

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

**8. PERFORMING ORGANIZATON REPORT NUMBER**

ISI/RR-94-393

**9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)**

Advanced Research Projects Agency / ASTO
3701 N. Fairfax Drive
Arlington, VA 22203-1714

Naval Research Laboratory
4555 Overlook Ave., SW
Washington, D.C. 20375-5000

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

**12A. DISTRIBUTION/AVAILABILITY STATEMENT**

UNCLASSIFIED/UNLIMITED

**12B. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

In a dynamic, multi-agent environment, an automated intelligent agent is often faced with the possibility that other agents may instigate events that hinder or help the achievement of its own goals. To act intelligently in such an environment, an automated agent needs an event tracking capability to continually monitor the occurrence of such events and the temporal relationships among them. This capability enables an agent to infer the occurrence of important unobserved events as well as to obtain a better understanding of the interaction among events. This paper focuses on event tracking in one complex and dynamic multi-agent environment: the air-combat simulation environment. It analyzes the challenges that an automated pilot agent must face when tracking events in this environment. This analysis reveals three new issues that have not been addressed in previous work in this area: (i) tracking events generated by agents' flexible and reactive behaviors, (ii) tracking events in the context of continuous agent interactions, and (iii) tracking events in real-time. The paper proposes one solution to address these issues. A key idea in this solution is that the mechanisms that an agent employs in generating its own flexible and reactive behaviors can be used to track other agents' flexible and reactive behaviors in real-time. The solution is demonstrated using an implementation of an automated pilot agent.

**14. SUBJECT TERMS**

Agent modeling, event tracking, multi-agent systems, plan recognition, real time

**15. NUMBER OF PAGES**

28

**16. PRICE CODE**

| 17. SECURITY CLASSIFICTION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UNLIMITED |

## GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

| | |
|---|---|
| C - Contract | PR - Project |
| G - Grant | TA - Task |
| PE - Program Element | WU - Work Unit Accession No. |

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

| | |
|---|---|
| DOD | - See DoDD 5230.24, "Distribution Statements on Technical Documents." |
| DOE | - See authorities. |
| NASA | - See Handbook NHB 2200.2. |
| NTIS | - Leave blank. |

Block 12b. Distribution Code.

| | |
|---|---|
| DOD | - Leave blank. |
| DOE | - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports. |
| NASA | - Leave blank. |
| NTIS | - Leave blank. |

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

# Event Tracking in a Dynamic Multi-agent Environment

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

11 September 1994

## Abstract

In a dynamic, multi-agent environment, an automated intelligent agent is often faced with the possibility that other agents may instigate events that hinder or help the achievement of its own goals. To act intelligently in such an environment, an automated agent needs an *event tracking* capability to continually monitor the occurrence of such events and the temporal relationships among them. This capability enables an agent to infer the occurrence of important unobserved events as well as to obtain a better understanding of the interaction among events. This paper focuses on event tracking in one complex and dynamic multi-agent environment: the air-combat simulation environment. It analyzes the challenges that an automated pilot agent must face when tracking events in this environment. This analysis reveals three new issues that have not been addressed in previous work in this area: (i) tracking events generated by agents' flexible and reactive behaviors, (ii) tracking events in the context of continuous agent interactions, and (iii) tracking events in real-time. The paper proposes one solution to address these issues. A key idea in this solution is that the mechanisms that an agent employs in generating its own flexible and reactive behaviors can be used to track other agents' flexible and reactive behaviors in real-time. The solution is demonstrated using an implementation of an automated pilot agent.

**Keywords**: Event tracking, real-time, plan recognition, agent modeling, multi-agent systems

# 1. Introduction

An automated intelligent agent pursuing its goals in a dynamic, multi-agent environment often encounters a large number of events that significantly impact the actions it needs to take to achieve its goals. Some of these events may be instigated by the agent itself. Others may be instigated by other agents as they pursue their own goals, which may conflict or coincide with the goals of this agent. As time marches on, these events rapidly continue to unfold.

To achieve its goals in such an environment, the automated agent needs to monitor both the occurrence of events in its world and the temporal relationships among them (e.g., the particular sequence in which they occur). This information is essential for reacting intelligently to the on-going events, particularly since it enables an agent to infer the occurrence of important unobserved events. Consider the following example from the domain of simulated tactical air-combat [Jones et al. 93, Tambe et al. 94]. This domain is based on a "real-world" simulator that has been commercially developed for the military [Calder et al. 93]. The automated agents are to act as automated pilots for the simulated aircraft in this domain. These automated pilots will take part in exercises with human fighter pilots, where they will aid in tactics development and training. For effective performance in this domain, these automated pilots must, among other things, continually monitor events in their environment. For instance, one crucial event is an opponent's firing a missile at an automated pilot's aircraft, threatening its very survival. Yet, the automated pilot cannot directly see the missile until it is too late to evade it. Fortunately, the automated pilot can monitor the opponent's sequence of maneuvers, and infer the possibility of a missile firing based on them, as shown in Figure 1. The automated pilot is in the dark-shaded aircraft and its opponent in the light-shaded one.

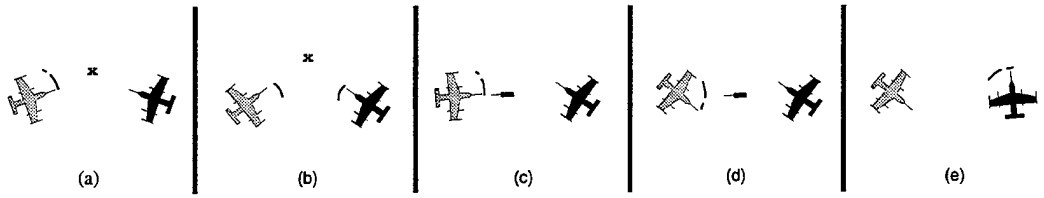



**Figure 1:** A portion of an air-combat simulation scenario: the automated pilot is in the dark-shaded aircraft and its opponent in the light-shaded one.

Suppose, initially, the two aircraft are positioned as shown in Figure 1-a. The range (distance) between the two aircraft is more than 10-15 miles, so they can only see each other on radar. The two are headed on a collision course, i.e., if they continue to fly straight, they will collide at the point shown by x. In the figure, a dashed line across an aircraft's nose indicates its direction of turn. Thus, in Figure 1-a, the opponent has just turned her aircraft to be on the collision course. Suppose the automated pilot observes this turn on its radar. Given that the collision course maneuver is often used as the quickest way to get to one's missile firing range, the automated pilot can infer that the opponent is possibly attempting to reach her missile firing range to fire a missile at it. In Figure 1-b, the automated pilot turns her aircraft, fifteen degrees to its right. The opponent responds in kind by turning her aircraft fifteen degrees to her left, so as to remain on collision course. In Figure 1-c, the opponent reaches her missile range. Here, she turns her aircraft to point straight at the automated pilot's aircraft and then fires a radar-guided missile, shown by -.

While the automated pilot can not observe this missile, based on its observation of the opponent's turn, it can infer that the opponent may be pointing at it as part of her missile firing behavior. Unfortunately, at this point, it cannot be certain about the opponent's missile firing, at least not to an extent where trained fighter pilots would infer a missile firing. However, if the opponent subsequently executes an *Fpole*

maneuver then that considerably increases the likelihood of a missile firing. This maneuver involves a 25-50 degree turn away from the automated pilot's aircraft, as shown in Figure 1-d (this maneuver is executed after firing a missile to provide radar guidance to the missile, while slowing the closure between the two aircraft). While at this point the opponent's missile firing is still not an absolute certainty, its likelihood is high enough, so that trained fighter pilots react as though a missile has actually been fired. The automated pilot reacts in a similar manner, by engaging in a missile-evasion maneuver. This involves turning the aircraft roughly perpendicular to the missile-flight (Figure 1-e), which causes the aircraft to "drop-off" (become invisible to) the opponent's radar. Deprived of radar guidance, the opponent's missile is rendered harmless.

The above example illustrates that an automated pilot needs to continually monitor events in its world, such as the opponent's turns and her (inferred) missile firing behavior, so as to react to them appropriately. In addition, it is important for the automated pilot to record the temporal relationship among these events — these relationships are key in inferring the occurrence of unobserved events such as the missile firing behavior. In the above example, if the opponent had first performed an Fpole maneuver, and later turned to collision course to get to her missile firing position, inferring her missile firing behavior would have been inappropriate.

We refer to this capability of monitoring events and their temporal relationships as *event tracking*. An event here may be considered as any coherent activity over an interval of time. This event may be a low-level action, such as an agent's Fpole turn, or it may be a high-level behavior, such as its missile-firing behavior, possibly inferred from a sequence of such turns. The event may be internal to an agent, such as maintaining a goal or executing a plan, or external to it, such as executing an action. The event may be observed by an agent, or simply inferred. The event may be instigated by any of the agents in the environment, including the agent tracking the events, or by none of them (e.g., a lightning bolt). *Tracking* any one of these events refers to recording that event in memory, recording the temporal relationship of that event with other events, and monitoring the progress of that event.

To understand the above in more detail, it is useful to view events and event tracking from another perspective. In particular, an event $\boldsymbol{E}$ may be defined as consisting of a set of sub-events $\{E_1, E_2,...E_N\}$ and a set $\boldsymbol{R} = \{(E_i\ r_1\ E_j), (E_i\ r_2\ E_k)...\}$ of temporal relationships among the sub-events. The event $\boldsymbol{E}$ and its sub-events $E_1$, $E_2$, etc. may be any of the variety of event types mentioned earlier (higher/lower-level, observed/unobserved, etc.), while the temporal relationships in $\boldsymbol{R}$ may be any of the temporal relationships from Allen's interval algebra ("before", "after", "during", etc.) [Allen 83]. For instance, the event $\boldsymbol{E}$ may be the opponent's missile firing behavior, and the relevant subevents and their temporal relationships may be that the opponent first maneuvered to achieve her missile firing position, then turned to point her aircraft at the automated pilot, and followed that with an Fpole maneuver, as shown in Figure 1. Tracking $\boldsymbol{E}$ then refers to tracking the occurrence of its sub-events and checking if the relationships $\boldsymbol{R}$ hold among these sub-events.

Note that, in general, an event $\boldsymbol{E}$ or its sub-events need not all be instigated by a single agent. For example, consider a situation where there are two opponents attacking the automated pilot's aircraft, as shown in Figure 2-a. Again, the automated pilot is in the dark-shaded aircraft, and the opponents are in the light-shaded aircraft. These opponents are closely coordinating their attack. One method of such close coordination is shown in Figure 2-b. Here, the opponent closer to the automated pilot's aircraft (the *lead*) leads this attack, while the second opponent, marked with **x** (the *wingman*) just stays close to the lead, and follows her commands. Thus, as the lead turns to gain positional advantage, the wingman turns in that direction as well, so as to stay close to the lead. A second method of close coordination is shown in

Figure 2-c. Here, the opponents execute a coordinated *pincer* maneuver — as the lead turns in one direction, the wingman turns in the opposite direction, so as to confuse the automated pilot and attack it from two sides. Other maneuvers involving close coordination between the two opponents are also possible. Essentially, these coordinated maneuvers are events where the sub-events are not instigated by a single agent, e.g., a pincer event consists of two opponents simultaneously turning in opposite directions around the automated pilot's aircraft.

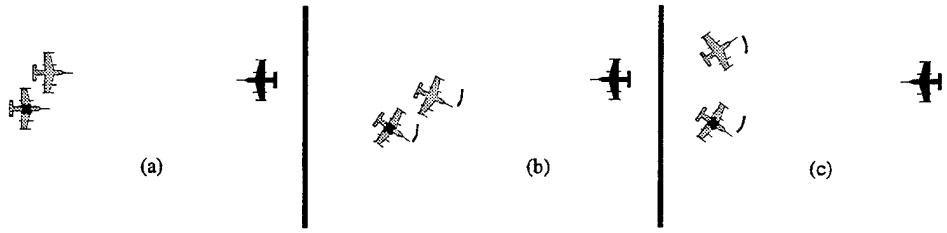



**Figure 2:** Events involving multiple agents: (a) two opponents attacking the automated pilot's aircraft; (b) opponents stay close; (c) opponents stage a coordinated "pincer".

Finally, an event may also involve sub-events instigated by the automated pilot itself. For instance, it is crucial for an automated pilot to check if it fired its missile at the opponent before the opponent fired a missile at it, or if the two were fired simultaneously, or if the opponent managed to fire her missile first. Here, the individual missile firings are sub-events of a higher-level event that tracks the temporal relationships among them. Tracking such events enables an agent to obtain a better understanding of its interactions with other agents, so as to react to them appropriately. In this case, tracking the order of missile firings enables an automated pilot to determine if it should engage in a missile-evasion maneuver (it may not engage in this maneuver if it fired its missile first).

Event tracking is closely related to the problem of plan or situation recognition [Kautz and Allen 86, Song and Cohen 91, Dousson et al. 93], the process of inferring an agent's plan (or a situation) based on observations of the agent's actions and the temporal relationships among those actions. The term event tracking is preferred in this investigation, since it also involves events other than plans, and since it is a continuous on-going activity, rather than one-shot recognition. However, more important than the terminology, of course, is gaining a better understanding of the nature of this capability. In particular, does the realistic multi-agent setting of air-combat simulation reveal anything new about event tracking? Given the complexity of this domain, answering this question in its entirety is beyond the scope of this single investigation. However, this paper takes a first step by focusing on the actions and behaviors of one or two opponents as they confront the automated pilot agent (henceforth referred to as AP). Section 2 illustrates that even within this restricted context, the air-combat domain brings forth three novel issues in event tracking: (i) tracking agents' flexible and reactive behaviors; (ii) tracking events in the context of continuous agent interactions; (iii) tracking events in real-time.

Section 3 presents the basics of an approach that attempts to address these issues. The key idea here is that the mechanisms that an agent employs in generating its own flexible and reactive behaviors can be used to track other agents' flexible and reactive behaviors. Section 4 presents important refinements to this basic idea that address agent interactions, and enable real-time event tracking. The resulting approach is demonstrated using an implementation of an automated pilot agent for air-combat simulation. This automated pilot is based on a system called TacAir-Soar [Jones et al. 93, Tambe et al. 94], which has been developed within Soar, an integrated problem-solving and learning architecture [Laird, Newell, and Rosenbloom 87, Rosenbloom, et al. 91]. This implementation and its results are discussed in Section 5.

Finally, Section 6 presents a summary and issues for future work.

## 2. Event Tracking in Air-combat Simulation

In air-combat simulation, the key issue in event tracking arises from the fact that this is a dynamic environment involving continuous agent interactions. Agents in this environment cannot rigidly commit to fixed sequences of actions. Instead, they need high behavioral flexibility and reactivity in order to achieve their goals. For instance, in Figure 1-b, the opponent has to re-orient herself to a new collision course heading in response to AP's turn. If AP had turned in the opposite direction, so would have the opponent. A more complex interaction occurs in Figure 1-e, where AP's missile evasion maneuver is a response to the opponent's overall maneuvers in Figures 1-a through 1-d, which are identified as part of her missile firing behavior. If AP had executed this missile evasion maneuver in Figure 1-c, the opponent would have had to abandon her missile firing maneuvers. This agent interaction also extends to situations involving more than two aircraft. For instance, in the situation shown in Figure 2, all three agents — AP, the lead and the wingman — continually influence each other's actions and behaviors. If a fourth aircraft joins this combat situation, then it will also interact with the other aircraft involved in the combat.

This dynamic interaction among the agents leads to the primary issue in event tracking in this domain: an agent must be able to track highly flexible and reactive behaviors of its opponent(s). A second closely related issue here is that while tracking its opponents' behaviors, an agent must also keep track of the appropriate agent interactions. These agents are not acting in a vacuum, but rather, they are continuously influencing each other. Without an understanding of these interactions, an opponent's actions may lead to unuseful or even misleading interpretations. For instance, the opponent's turn in Figure 1-b needs to be tracked as a response to AP's own turn. Otherwise, the opponent's turn may appear meaningless. In other situations, if the opponent engages in the missile-evasion maneuver, it is important to track that as a response to AP's missile firing behavior. Similarly, in Figure 2, it is important to understand the interaction between the lead and the wingman, to track the wingman's actions.

A third related issue here is that event tracking must occur in real-time, and must not hinder an agent from acting in real-time. For instance, in Figure 1, if AP does not track the missile firing in real-time or does not react to it in real-time, the results could be fatal.

A fourth issue relates to tracking events in the presence of ambiguity in an opponent's behaviors. For instance, consider the opponent's turn in Figure 1-c. From AP's perspective, there is considerable ambiguity about the opponent's exact maneuver. Clearly, the opponent could be turning to fire a missile. However, it is also possible that she has decided to run away, and as part of that, she is making a big 180-degree turn, and AP is simply observing the initial portion of that turn. Finally, it is also possible that due to a glitch in her radar apparatus, she has lost radar contact with AP, and she is maneuvering to try to regain radar contact. AP must be able to deal with at least three aspects of such ambiguous situations. First, AP must be able to represent relevant aspects of such a situation, so that it can continue to track events as the situation changes. For instance, as the opponent continues her maneuvers, turning as shown in Figure 1-d, AP should be able to continue to track her maneuvers. Second, AP should be able to react appropriately when faced with an ambiguous situation. For instance, the ambiguity in the opponent's maneuvers in Figure 1-c makes it problematic for AP to choose an appropriate reaction: should it chase the opponent, or should it prepare for a missile evasion maneuver? AP should be able react appropriately in such problematic situations. Third, AP should not incur large overheads or delays in dealing with such ambiguous situations (e.g., in processing their representations), so as to hinder its own real-time performance. This last aspect of dealing with ambiguity illustrates its strong interaction with the issue of

real-time event tracking.

Thus, this domain raises a challenging combination of issues for event tracking. Except for the issue of ambiguity, all of the issues presented are novel ones. In particular, in previous investigations in the related areas of plan/situation recognition [Kautz and Allen 86, Song and Cohen 91, Dousson et al. 93, Van Beek and Cohen 93, Carberry 90a] — including one investigation focused on plan recognition in airborne tactical decision making [Azarewicz, et al. 86] — these issues have not been addressed. With regard to the first two issues, plan recognition models have not been applied in such dynamic, interactive multi-agent situations, and hence do not address strong interactions among agents or the resulting flexibility and reactivity in agent behaviors. In particular, these models assume that a single planning agent (or multiple independent planning agents) has some plans, and a recognizing agent recognizes these plans. The planning agent may be either actively cooperative (it intends for its plans to be recognized by the recognizing agent) or passive (it is unconcerned about its plans being recognized) [Carberry 90b]. The recognizing agent's job is to recognize these plans and possibly provide a helpful response. However, neither the recognizing agent, nor any other agents in the environment are assumed to have any influence on these plans. Consequently, these plan recognition models can rely on pre-compiled *plan libraries*, where each plan lists the sequence of events and the temporal relationships among the events [Song and Cohen 91]. The representation of a plan in such a library is similar to the explicit representation of an event in terms of an enumeration of its subevents and their temporal relationships. A plan library represents the entire class of events of interest in this fashion. However, as indicated by the previous discussion in this section, it is difficult to employ such libraries in tracking agent behaviors in the air-combat simulation environment. In particular, given the agents' high behavioral flexibility and reactivity, all possible variations on agent behaviors would need to be included in such libraries, leading to a combinatorial explosion in the number of plans/events. In other words, while a specific event instance can be represented by enumerating its subevents and their temporal relationships, such a scheme is not sufficiently expressive to provide a compact representation for the entire class of events of interest. The approach presented in this article avoids such an enumeration.

With regard to the issue of tracking events in real-time, this has not been the focus of previous work in plan/situation recognition. There is some work that is beginning to address this issue [Dousson et al. 93], although it does not as yet deal with complex agent behaviors.

Thus, researchers working on plan-recognition have not dealt with such interactive, real-time, multi-agent situations. There are, however, two other sub-areas of AI that have dealt with such situations. The first such sub-area is Distributed AI (DAI). There is some work in DAI on understanding other agents' plans (e.g., see [Durfee and Lesser 88]). However, it focuses on agents exchanging their plan data structures for active cooperation, rather than on plan recognition. The second sub-area is game playing. Game trees clearly deal with interactive situations, and they also model an opponent's actions in such situations. Indeed, mini-max game tree search has been used to implement automated pilots for air-combat simulation [Katz 93, Schaper et al. 94]. However, there are several idealizations in these game-tree implementations. For instance, pilots are modeled as engaging in discrete turn by turn actions, as opposed to overlapping and continuous actions. Pilots are also assumed to have perfect information regarding the state of the "game", as opposed to the incomplete information resulting from their imperfect sensors. An aircraft's complex aero-dynamic movements, which constitute individual moves in the game tree, are modeled in a simplified manner to avoid significant computational overheads during look-ahead search. To avoid similar problems with modeling complex missile aero-dynamics, only a simple gun is modeled. (A detailed list of idealizations appears in [Katz 93].) Given these idealizations, game-tree techniques appear impractical, at least at present, for event tracking in air-combat simulations.

## 3. A Solution for Event Tracking

The proposed solution for event tracking is based on a core idea that addresses the issue of tracking flexible and reactive behaviors. This idea is explained in this section. Important refinements, that address the other issues in event tracking, will be discussed in the next section.

The core idea is based on the following observation. All of the agents in this environment possess similar flexibility and reactivity in their behaviors. In particular, AP shares this similarity with its opponent. Thus, the mechanisms that AP employs in generating flexible and reactive behaviors may be used in service of tracking flexible and reactive behaviors of other agents. To understand this idea in detail, it is first useful to understand the mechanisms that AP employs to generate its own flexible and reactive behaviors. Section 3.1 explains this using a concrete implementation of AP in TacAir-Soar. As mentioned earlier, the Soar architecture forms the basis of TacAir-Soar. For the purposes of the following description, we need to focus only on one important aspect of the Soar architecture — its problem space model of problem solving. Very briefly, a problem space consist of states and operators. An agent solves problems in a problem space by taking steps through it to reach a goal. A step in a problem space usually involves applying an operator in the problem space to a state. This operator application changes the state. If the state changes caused by the application of the operator, or by new inputs received via the agent's sensors, satisfy the operator's termination conditions, then that operator is terminated, and a new operator is applied. If the termination conditions remain unsatisfied, then a subgoal is created, where a new problem space is installed to attempt to make progress in problem solving. By subgoaling from one problem space into another, a whole goal/problem-space hierarchy may be generated.

Following this illustration of AP's flexible and reactive behaviors, Section 3.2 explains how it may be exploited for tracking other agent's behaviors.

### 3.1. An Agent's Own Behavior

Figure 3 illustrates the problem spaces and operators that AP employs while it is trying to get into position to fire a missile. In the figure, problem spaces are indicated by boxes, with problem-space names indicated by bold letters outside the boxes. Operators in problem spaces are indicated by the text inside the boxes. The operators that are currently being applied are shown in italics. For example, *INTERCEPT*, an operator for engaging in combat with enemy aircraft, is currently being applied in the **EXECUTE-MISSION** problem space. The other operators in a problem space are alternatives to the operator currently being applied (these are un-italicized). These alternatives are not being used since either they are inapplicable in the current situation, or they are less preferable than the operator currently being applied. For example, consider the FLY-RACETRACK operator in the **EXECUTE-MISSION** problem space. This involves flying in a racetrack pattern searching for enemy aircraft when there are no enemy aircraft present. However, FLY-RACETRACK is currently inapplicable, since an enemy aircraft is present.

The operator hierarchy in Figure 3 is generated as follows. In the top-most problem space, **TOP-PS**, AP is attempting to execute its mission by applying the *EXECUTE-MISSION* operator. This is the only operator it has in this problem space. The termination condition of this operator is the completion of AP's mission, which may be, for example, to protect its home-base for a specific time period. Since this is not yet achieved, a subgoal is generated. This subgoal involves the **EXECUTE-MISSION** problem space. Here, AP employs the *INTERCEPT* operator. However, the termination condition of this operator — the opponents are either destroyed or chased away — is not yet achieved. This leads to a subgoal into the **INTERCEPT** problem space, where AP applies the *EMPLOY-MISSILE* operator. However, the missile

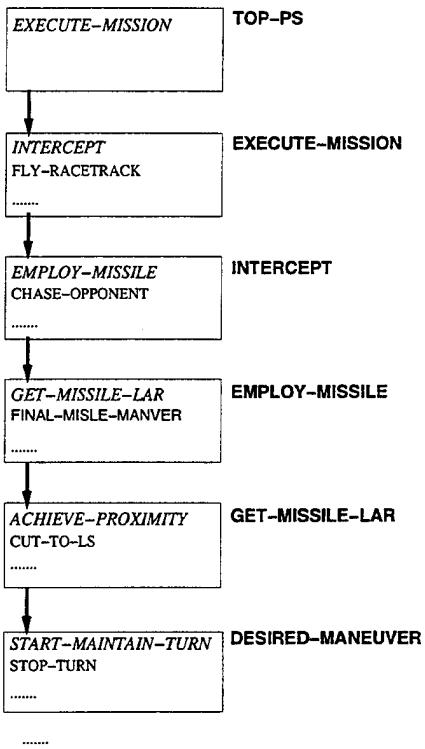



**Figure 3:** AP's problem space/operator hierarchy.

firing range and position is not yet reached. Therefore, AP applies the *GET-MISSILE-LAR* operator in a subgoal. (LAR stands for launch-acceptability-region, the region from where AP may fire a missile at its opponent). This subgoaling continues until the application of the *START-AND-MAINTAIN-TURN* operator in the **DESIRED-MANEUVER** problem space, which causes AP to turn. Later, the *STOP-TURN* operator is applied to stop the aircraft's turn when it reaches its required heading.

This operator organization supports AP's flexible and reactive behaviors, given appropriate pre-conditions (or conditions) for the operators, and the appropriate operator selection and termination mechanisms [Laird and Rosenbloom 90]. For instance, there is a global state shared by all of the problem spaces. If this state changes so that the termination conditions of any of the operators in the operator hierarchy are achieved, then that operator can be terminated. All of the subgoals generated due to that operator are then automatically deleted. Thus, if the opponent simply begins to run away as AP is attempting to apply the *EMPLOY-MISSILE* operator, then this operator may be terminated, and its subgoals will be automatically deleted. The *CHASE-OPPONENT* operator may be selected instead. There are other mechanisms adding to AP's reactivity as well [Laird and Rosenbloom 90].

Since all of the above operators are used in the generation of AP's own actions, they will be henceforth denoted using the subscript *own*. For instance, $EMPLOY\text{-}MISSILE_{own}$ will denote the operator AP uses in employing a missile. $Operator_{own}$ will be used to denote a generic operator that AP uses to generate its own actions. The global state in these problem spaces will be denoted by $state_{own}$. Problem-spaces that consist of $state_{own}$ and $operator_{own}$ will be referred to as *self-centered* problem spaces. The motivation for using this method for denoting states, operators and problem spaces will become clearer below.

### 3.2. Tracking Other Agent's Behaviors

Given the similarity of AP and its opponent's flexible and reactive behavior, the key idea is to use the mechanism that AP uses for its own behavior to track its opponent's behaviors. We will first illustrate this idea in some detail using some simplifying assumptions. The assumptions will be addressed in more detail in the next section.

We begin with the simplifying assumption that AP is engaged in a combat with only a single opponent. Additionally, we assume that AP and its opponent are exactly identical in terms of the problem spaces and operators they employ to engage in simulated air-combat. Thus, AP can essentially use a copy of its self-centered problem spaces to track the opponent's actions and behaviors. We will refer to these copies as *opponent-centered* problem spaces. Operators in these problem spaces represent AP's model of its opponent's operators. These operators are denoted using the subscript *opponent*. Thus, the $EXECUTE\text{-}MISSION_{opponent}$ operator is used in modeling an opponent's execution of her mission. Similarly, $operator_{opponent}$ is used to model a generic operator used by the opponent. The global state in these problem spaces represents AP's model of the state of its opponent, and is denoted by $state_{opponent}$. Clearly, it is not straightforward for AP to create a $state_{opponent}$ that accurately models the opponent's state. Essentially, it requires mirroring all of the information on AP's $state_{own}$. For instance, $state_{opponent}$ requires information that the opponent is likely to obtain from her radar regarding AP's aircraft, such as the heading and altitude of AP's aircraft, the range (distance) between AP's aircraft and her aircraft, the *target aspect* from her perspective (the angle between AP's straight line flight path and the opponent's position), the *angle off* from her perspective (the angle between the opponent's flight path and AP's position) and so on. All of these are important quantities, since many of the opponent's actions depend on them. Yet their calculations can be very expensive. Furthermore, these quantities have to be continuously re-calculated, since they are dependent on the aircraft positions, which are continuously changing. There are other aspects of $state_{opponent}$, such as information regarding the opponent's mission, that are also difficult to generate. So another simplifying assumption that we make here is that AP has already generated an accurate $state_{opponent}$.

With the opponent-centered problem spaces, $state_{opponent}$ and $operator_{opponent}$, AP can pretend to be in the opponent's position. AP then tracks the opponent's actions and behaviors by pretending to engage in the same actions and behaviors as the opponent. For instance, to track the opponent's turn towards a particular direction, AP applies $START\text{-}AND\text{-}MAINTAIN\text{-}TURN_{opponent}$ to $state_{opponent}$. This operator application does not directly change $state_{opponent}$ — it is not an actual implementation of the turn, but rather, it is AP's model of the opponent's turning action, essentially used for recognizing that action. Thus, if the opponent actually starts turning in the direction indicated by $START\text{-}AND\text{-}MAINTAIN\text{-}TURN_{opponent}$, then AP considers its model of the opponent's action corroborated. The change in the opponent's heading due to its turn is then noted in $state_{opponent}$.

This technique of event tracking, where an agent models another by pretending to be in that agent's position, has been previously used in automated tutoring systems [Anderson, et al. 90, Ward 91]. These tutoring systems need the ability to model the actions of the students being tutored. For this, these systems use student-centered problem spaces in which states and operators model the students under scrutiny. This student modeling technique is referred to as *model tracing*. The approach proposed here for event tracking is thus based on this model tracing work. However, there are some significant differences. For example, previous work has primarily focused on static, single-agent environments, where the agent being modeled is the only one causing changes in the environment [Hill and Johnson 93]. However, before exploring these differences further, it is first useful to understand AP's event tracking in more detail. This is explained below, using the illustration of the multiple problem space hierarchies in Figure 4. Note that

while this explanation does not directly describe the operation of an actual implementation, it is based on an actual implementation that will be described in Section 5. Basically, the description presented here illustrates the core idea, and further modifications described in later sections lead up to the implementation described in Section 5.

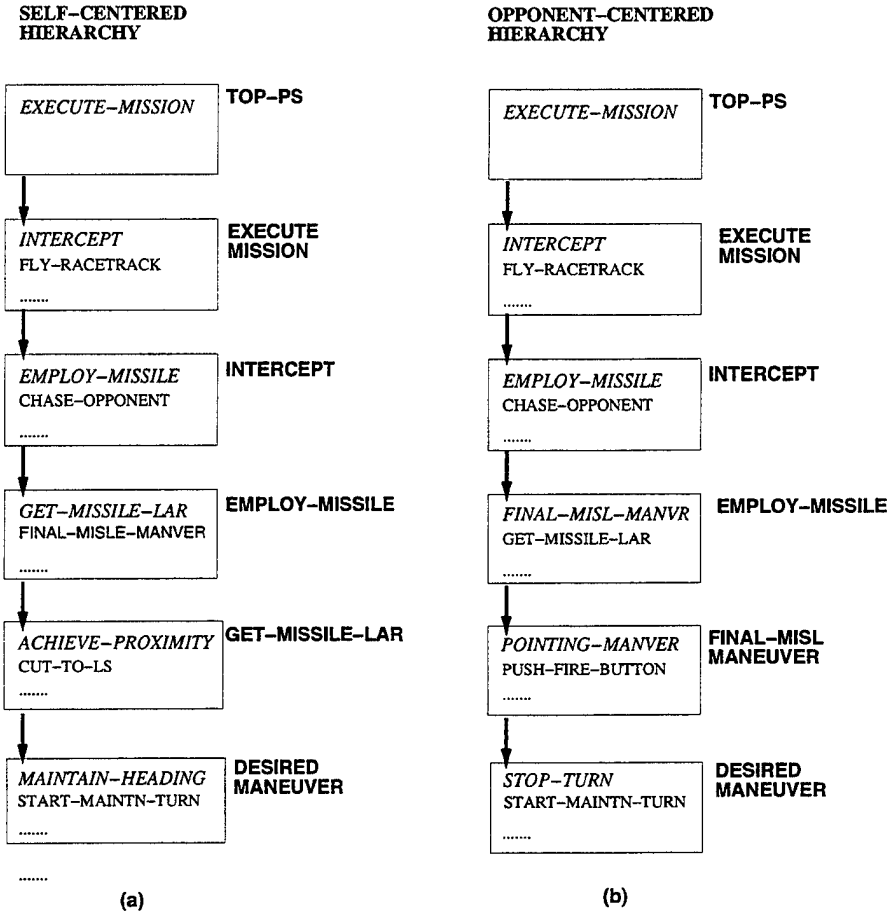



**Figure 4:** (a) AP's self-centered operator hierarchy, (b) AP's model of opponent's operator hierarchy.

Figure 4 shows two problem space hierarchies. AP executes these hierarchies in parallel in order to engage in air-combat simulation. The first hierarchy, shown in Figure 4-a, contains self-centered problem spaces, and allows AP to generate its own behavior. The second hierarchy, shown in Figure 4-b, contains opponent-centered problem spaces. AP is using this hierarchy to track the opponent's behaviors in the situation illustrated in Figure 1-c. Here, the *EXECUTE-MISSION*$_{\text{opponent}}$ operator models the opponent's execution of her mission. Since the opponent's mission is not completed, a subgoal is generated, within which the operator *INTERCEPT*$_{\text{opponent}}$ is applied. This operator subgoals into the **EMPLOY-MISSILE**$_{\text{opponent}}$ problem space. Here, the *FINAL-MISSILE-MANEUVER*$_{\text{opponent}}$ operator models the the opponent's final missile launching behavior. Again, this operator subgoals, and the operator *POINTING-MANEUVER*$_{\text{opponent}}$ is applied. This operator finally subgoals into **DESIRED-MANEUVER** problem space, where the *START-AND-MAINTAIN-TURN*$_{\text{opponent}}$ operator is applied. If the opponent actually starts turning towards AP's aircraft, then the operator *START-AND-MAINTAIN-TURN*$_{\text{opponent}}$ is corroborated. AP relies on this corroboration to corroborate the other current operators — such as *FINAL-MISSILE-MANEUVER*$_{\text{opponent}}$ and *POINTING-MANEUVER*$_{\text{opponent}}$ — in its opponent-centered problem space hierarchy. Essentially, AP's

model of its opponent is that the opponent is engaging in final missile maneuvers, and as part of that, she is turning to point at AP's aircraft. As discussed in the previous section, in reality, there is some ambiguity about the opponent's exact maneuver. If the opponent is actually running away, then AP's model of its opponent's operator hierarchy is inaccurate. This issue will be addressed in the next section.

Continuing with the example in Figure 4-b, when the opponent's aircraft turns sufficiently to point straight at AP's aircraft as shown in Figure 1-c, the termination condition of the *POINTING-MANEUVER*$_{opponent}$ operator is satisfied, and this operator is terminated. A new operator from the **FINAL-MISSILE-MANEUVERS**$_{opponent}$ problem space — *PUSH-FIRE-BUTTON*$_{opponent}$ — is now applied. This operator predicts a missile firing, but it is known that that cannot be observed. Hence, *PUSH-FIRE-BUTTON*$_{opponent}$ is terminated even though there is no direct observation to support that termination. (This corroboration without observation requires the addition of some knowledge that is specific to opponent-centered problem spaces, i.e., not copied from self-centered problem spaces.) This also corroborates and terminates the *FINAL-MISSILE-MANEUVERS*$_{opponent}$ operator. However, the resulting missile firing is marked as not being highly likely. Following that, an *FPOLE*$_{opponent}$ operator in the **EMPLOY-MISSILE**$_{opponent}$ problem space predicts an Fpole turn. When the opponent executes her Fpole turn in Figure 1-d, the *FPOLE*$_{opponent}$ operator is corroborated and terminated. With this Fpole turn, the missile firing is now considered as being highly likely. AP may now attempt to evade the missile.

Thus, AP uses a uniform mechanism for generating flexible and reactive behaviors as well as for tracking them. This uniformity is the core idea that enables AP to track flexible/reactive behaviors. In particular, operator$_{opponent}$ can be activated and terminated in the same flexible manner as operator$_{own}$. Other mechanisms of reactivity available to AP in generating its own behavior are also available to it in modeling the opponent's reactive behavior. Viewing this from another perspective, AP does not track events by explicitly pre-enumerating all possible chains of sub-events. Instead, it relies on dynamic generation of the sub-events. Essentially, each operator$_{opponent}$ represents an event ***E***. The sub-operators of operator$_{opponent}$ — those applied in a subgoal of operator$_{opponent}$ — correspond to the sub-events of ***E***, i.e., the set $\{E_1, E_2..E_N\}$. The sequence of sub-operator execution corresponds to the temporal relationships ***R*** among the sub-events. Thus, the application of operator$_{opponent}$ corresponds to tracking an event ***E*** by dynamically generating its sub-events.

While the above approach addresses the issue of tracking flexible and reactive behaviors, it does not address the other important issues — real-time, agent interactions, and ambiguity — raised in Section 2. In addition, the approach concentrates only on events instigated by a single opponent, and does not enable AP to track events instigated by multiple agents, such as a pincer. This concern is related to the first simplifying assumptions made earlier in this section, that there is only a single opponent in the world. Two other assumptions were also made: (i) AP and the opponent are identical in terms of the problem-spaces and operators; and (ii) AP can accurately generate state$_{opponent}$. All of these issues and assumptions will be discussed further in the next section.

## 4. Addressing Issues in Event Tracking

This section presents some important refinements to our event tracking approach that enable it to address the issues and assumptions mentioned in the previous section. Subsection 4.1 presents some refinements that address the issue of ambiguity in event tracking. Subsection 4.2 addresses the issue of real-time event tracking. Subsection 4.3 addresses the issue of agent interactions. Finally, Subsection 4.4 addresses the three assumptions mentioned at the end of the previous section. The result is a robust

approach for event tracking, and it is presented in an algorithmic form in Section 4.5.

## 4.1. Ambiguity in Event Tracking

As mentioned in Section 2, there are three aspects to dealing with ambiguous situations in event tracking: (i) representation of ambiguous situations; (ii) facilitation of an appropriate reaction; and (iii) avoidance of overheads that can hinder real-time tracking. Suggested solutions for addressing ambiguity typically address the representational aspect of this issue, but not the other two. In particular, one method of addressing the ambiguity is to maintain multiple operator hierarchies, so as to track each possibility independently [Ward 91]. Thus, if there is some ambiguity about whether the opponent is actually executing $RUN\text{-}AWAY_{opponent}$ or $EMPLOY\text{-}MISSILE_{opponent}$, then there are two separate operator hierarchies maintained for the two possibilities. If there is further ambiguity in the $EMPLOY\text{-}MISSILE_{opponent}$ possibility between $GET\text{-}MISSILE\text{-}LAR_{opponent}$ and $POINTING\text{-}MANEUVER_{opponent}$ operators, then there are two additional operator hierarchies maintained for the two new possibilities. Thus, this method can represent all of the possibilities in an ambiguous situation. However, the multiple operator hierarchies imply that there are multiple interpretations for the opponent's actions: the opponent is possibly running away or possibly engaging in a missile firing behavior. Choosing an appropriate reaction is thus problematical for AP: should it chase the opponent, or should it prepare for a missile evasion maneuver? In addition, this method can potentially lead to a large number of operator hierarchies. Maintaining all of them can be extremely expensive, making it difficult to continue to track events in real-time. Thus, this method cannot satisfactorily deal with all of the aspects of ambiguity.

A possible refinement of the above method can help improve its performance. In particular, AP can delay tracking the opponent's maneuver until the completion of her maneuver. For instance, if the opponent starts a turn, AP could maintain in memory information regarding the starting point of that turn, and then wait till the completion of the turn before tracking it. This can help reduce the ambiguity, and hence the number of operator hierarchies required to track the opponent's actions. However, such a delay in tracking the opponent's actions is sometimes unaffordable — it can make it difficult for AP to react appropriately, and allow the opponent to gain a positional advantage over AP. Additionally, in some cases, the opponent's aircraft may become invisible to AP's radar before her maneuver is completed. For instance, if the opponent performs a missile evasion maneuver, she disappears from AP's radar before the completion of her maneuver. Thus, it may not always be possible for AP to wait for the completion of the opponent's maneuver before tracking it.

To avoid these problems, AP relies on a single operator hierarchy to continually track the opponent's actions — there is a single on-going interpretation of the opponent's actions. This single interpretation enables AP to react appropriately to the opponent's on-going actions. This solution also does not burden AP with processing multiple operator hierarchies, thus helping it to track events in real-time. However, the single operator hierarchy makes it difficult for AP to represent ambiguous situations. To alleviate this problem, AP uses two different strategies to attempt to resolve the ambiguity. The first, *passive ambiguity resolution* strategy is to rely on operator selection heuristics. The key heuristic here is the *worst case* assumption. In particular, given that this is an adversarial environment, AP may always select the worst-case operator (from its own perspective) while tracking the opponent's actions. Thus, if there is ambiguity between $RUN\text{-}AWAY_{opponent}$ or $EMPLOY\text{-}MISSILE_{opponent}$, AP will select $EMPLOY\text{-}MISSILE_{opponent}$ to track the opponent. With this heuristic, AP's reaction to an opponent's on-going action will always be a very cautious one. Indeed, AP's reaction may be overly cautious, and other strategies are required to resolve the ambiguity.

The second, more novel strategy is the *active ambiguity resolution* strategy. A "strong" version of this strategy is for AP to act in a such a way so as to force the opponent to resolve the ambiguities in its actions. For instance, AP can "fake" a missile firing by engaging in a deceptive Fpole maneuver, to check if the opponent is reacting to its actions — it is possible that the opponent has not yet seen AP on radar. AP may also rely on a "weak" version of the active resolution strategy. It may avoid actions that are likely to engender an ambiguous response. For instance, in some cases, AP may avoid turning while the opponent is also turning, since the opponent may or may not accurately observe such a turn on her radar, and thus her response may be ambiguous. In employing an active resolution strategy, AP's event tracking resembles an *active vision* system [Ballard 89], that can take actions, such as moving cameras to get closer to objects or changing focus, to simplify the computation required in early vision. Unfortunately, unlike an active vision system, AP cannot dedicate itself to active ambiguity resolution, since it also needs to accomplish its own goals, including getting into position to fire a missile at the opponent. Therefore, ideally, AP needs to execute maneuvers that accomplish its own goals, while simultaneously aiding it in active ambiguity resolution. However, discovering such maneuvers is extremely difficult, since deep domain knowledge is required for such a discovery. Therefore, AP currently does not discover new active resolution maneuvers, and instead relies on a fixed set of known maneuvers (supplied by the domain experts).

Thus, using its different ambiguity resolution strategies, AP attempts to use a single $\text{operator}_{\text{opponent}}$ hierarchy to track the opponent's actions and behaviors in real-time. However, despite this, AP cannot eliminate inaccuracies in its tracking. Such inaccuracies usually manifest themselves in terms of *match failures*. For instance, AP's passive ambiguity resolution heuristic may select the $\textit{POINTING-MANEUVER}_{\text{opponent}}$ operator for tracking the opponent's action. As mentioned earlier, this operator subgoals to where the $\textit{START-AND-MAINTAIN-TURN}_{\text{opponent}}$ is applied. For its corroboration, this operator requires that the opponent's aircraft turn towards AP's aircraft. However, in reality, the opponent may have decided to run away, possibly causing it to turn away from AP's aircraft. This difference in the anticipated and actual turn directions leads to a match failure, indicating an inaccuracy in tracking. Similar match failures can also occur if the opponent fails to stop turning even though a $\textit{STOP-TURN}_{\text{opponent}}$ anticipates the opponent's stopping its turn; or if the opponent starts turning even though $\textit{MAINTAIN-HEADING}_{\text{opponent}}$ anticipates the opponent to continue to maintain heading. Of course, in some cases, there may be an inaccuracy in tracking, and yet there may not be a match failure. For instance, in the above $\textit{POINTING-MANEUVER}_{\text{opponent}}$ case, the opponent may actually need to make a big 180-degree turn to run away. This may initially cause it to turn towards AP's aircraft, thus corroborating the $\textit{POINTING-MANEUVER}_{\text{opponent}}$ operator. However, in such cases, there is typically a match failure in some subsequent operator, such as $\textit{FPOLE}_{\text{opponent}}$. In all these cases of match failures, AP tries other operators from its opponent-centered problem spaces. Essentially, AP tries out a systematic backtrack search to attempt to locate an operator that can match the opponent's on-going action. We will return to this backtracking search in Section 4.5.

## 4.2. Real-time Event Tracking

AP faces two main obstacles when attempting to track events in real-time. The first is dealing with ambiguity, which we have discussed in detail in the previous section. The second is tracking an agent's detailed actions. As Hill and Johnson [Hill and Johnson 93] have recently argued, detailed actions may overwhelm an agent with tracking overheads. This can be particularly problematical for AP, since this may disallow it from acting in real-time. AP faces this problem due to the dynamic and realistic nature of its environment. In particular, there are almost always minor errors in the opponent's actions, or in her perception of AP's current position and heading. As a result, she makes continuous minor adjustments in

her own position and heading to compensate for her sensor and/or action errors. For instance, in Figure 1-a, the opponent can rarely achieve a precise collision course heading. As her aircraft continues to deviate from the collision course, she may then make a minor course correction, which may possibly over-compensate for the error and then cause her to turn again. There can also be some small two-three degree errors in AP's perception of the opponent's heading. If AP is to track the opponent's actions in detail, it can only do so by modeling these errors and corrective actions. The overheads of tracking at this level of detail can be overwhelming. Of course, for some maneuvers, such as the Fpole, the opponent may not be required to maintain a precise heading. In these situations, she may tolerate fairly large deviations (10 to 15 degrees) from her initial heading. Such maneuvers add to the tracking problem.

To avoid this problem, AP relies on filters that help it focus on *significant* events. One simple filter is to ignore any changes in the opponent's heading of less than three degrees, to compensate for the small errors in AP's perception of the opponent's heading. A second simple filter is the use of the *fuzz-box* approximation [McDermott and Davis 84]. A fuzz-box indicates AP's tolerance for deviations in a given quantity, such as heading, altitude, speed, etc. For instance, for the opponent's pointing maneuver, where the opponent points at AP to fire a missile, the fuzz-box accepts five degrees of deviation. Thus, as long as the opponent's heading is within five degrees of the heading required for the pointing maneuver, AP considers that as part of the opponent's pointing maneuver. While this small fuzz-box is sufficient for some maneuvers, maneuvers such as the Fpole discussed above, require a larger fuzz-box, accepting up to 20-degrees of deviation in heading. In general, these fuzz boxes may or may not be symmetrical. As of now, the fuzz box for each maneuver is hand-coded. Automatic discovery of the amount of "fuzziness" for each maneuver is a topic for future work.

### 4.3. Modeling Agent Interactions

The approach outlined in Section 3.2 requires AP to build multiple problem-space hierarchies, one for its own behaviors and one for modeling its opponent (and perhaps additional operator hierarchies for modeling other agents in the environment). To model real-world agent interactions, there is a need to facilitate communication among these problem space hierarchies. For the purpose of the following discussion, we may divide these agent interactions into three categories. The first category of interactions requires pre-conditions (or conditions) of operators in one hierarchy to reference the operators or the state from other hierarchies. For instance, to select $CHASE\text{-}OPPONENT_{own}$ operator, AP needs the ability to refer to the $RUN\text{-}AWAY_{opponent}$ operator from the opponent's operator hierarchy. Similarly, the $MISSILE\text{-}EVASION_{opponent}$ operator depends on the ability to reference $EMPLOY\text{-}MISSILE_{own}$ and vice versa. This first type of interaction implies that at the very least, the multiple problem space hierarchies cannot be watertight compartments, completely isolated from each other.

The second type of agent interaction requires that the continuously changing aircraft positions be communicated from one problem space hierarchy to another. For instance, suppose there is a change in AP's position and heading. This causes a change in $state_{own}$. This change has to be communicated to $state_{opponent}$ to keep it updated. Furthermore, given this change, quantities in $state_{opponent}$ that depend on AP's heading and position such as the range (distance) between the opponent and AP, their angle-off and target-aspect need to be re-calculated. Given the dynamic nature of the environment, where AP is continuously moving, and often turning its own aircraft, there is thus a need to almost continuously update much of the information on $state_{opponent}$ to keep it consistent. While $state_{own}$ has to be similarly updated after the opponent's moves, this updating is simplified since many of the calculations are automatically done by AP's radar. This form of interaction also includes the entities in $state_{own}$ and $state_{opponent}$ that represent the missiles that the agents fire at each other. In particular, if the opponent fires

a missile, some object or entity representing that missile needs to be communicated from $state_{opponent}$ to $state_{own}$, and vice versa.

Thus, this second form of agent interaction can involve very substantial communication and re-calculation overheads. In addition, it involves duplication of information — e.g., AP's heading is represented in both $state_{own}$ and $state_{opponent}$ — which can be problematical, as it may lead to inconsistencies. If there are other aircraft involved in the simulated combat, this duplication and overhead can be even more substantial. This can be particularly problematical for continued real-time event tracking.

The problem AP faces here is that opponent-centered and self-centered problem spaces are compartmentalized hierarchies. To avoid the duplications and expensive recalculations, one solution is to merge the different operator hierarchies into a single compartment, which we will refer to as a world-centered problem space (**WCPS** for short). A WCPS eliminates the boundaries between different self-centered and opponent-centered problem spaces. Instead, the different operator hierarchies are maintained within the context of a single WCPS, with a single world state. This single world state can now allow information sharing between $state_{own}$ and $state_{opponent}$, thus avoiding communication overheads and potential inconsistencies. For instance, AP's range to its opponent is identical to the opponent's range to AP. Thus, the range information, which AP has available on $state_{own}$ directly from its radar, can be directly used as the range in $state_{opponent}$. As this range changes in $state_{own}$, it is automatically updated in $state_{opponent}$. Thus, the range is shared among $state_{own}$ and $state_{opponent}$, avoiding recalculation. AP's heading in $state_{own}$ and $state_{opponent}$ can also be shared. Interestingly, the angle off in $state_{opponent}$ turns out to be the target aspect in $state_{own}$, and the target aspect in $state_{opponent}$ turns out to be the angle off in $state_{own}$. These quantities can also be shared, thus avoiding recalculations. Similarly, a single entity may be used to model a single missile in the world, thus avoiding duplicate representation of the missile. A WCPS encourages such sharing of information among the different self-centered and opponent-centered problem space hierarchies, and thus dramatically reduces the burden of modeling $state_{opponent}$. However, such sharing is clearly not always appropriate, e.g., AP may deceive the opponent into believing that it has fired a missile, without actually firing one. We will further discuss this issue in Section 6.

AP's shift from single-agent centered problem spaces to a WCPS also facilitates a third form of agent interaction. In particular, it enables AP to represent *multi-agent* operators that can track events instigated by multiple agents. For instance, consider a pincer event instigated by multiple opponents (See Figure 2). In a WCPS, this may be tracked by directly executing a multi-agent operator, i.e., $PINCER_{lead\text{-}and\text{-}wingman}$, which can operate on both on $state_{lead}$ and $state_{wingman}$, since both are part of a single WCPS. Similarly, a $FIRED\text{-}FIRST_{own\text{-}and\text{-}opponent}$ operator may be used to check if AP fired its missile before (or after) the opponent, which is again a multi-agent operator that can operate on both $state_{own}$ and $state_{opponent}$.

The shift from self-centered and opponent-centered problem-spaces to a WCPS is related to the *objective* framework used in simulation and analysis of DAI systems [Decker and Lesser 93], which describes the essential, "real" situation in the world. However, the focus of our work is on an individual agent using its own world-centered model for event-tracking. While this model introduces a shift towards an objective point of view, by definition, it is an agent's subjective view of its environment, and may contain approximations in $operator_{opponent}$ and $state_{opponent}$. The specific idea of avoiding duplicate representation of an object in $state_{own}$ and $state_{opponent}$, is similar to a key feature of the SNePS belief modeling framework [Shapiro and Rapaport 91]. In SNePS, the models of different agents' belief spaces may share the representation of an object and thus avoid duplication. WCPS extends this sharing to

dynamic environments, so as to avoid the cost of updates, and harnesses the sharing in service of event tracking.

### 4.4. Addressing Assumptions in Event Tracking

Three assumptions were outlined in Section 3.2. The first one was the presence of only a single opponent in the world. This assumption enabled AP to rely on a single $operator_{opponent}$ hierarchy to track its opponent's actions. If there are multiple opponents present, then in the simplest case, AP may track their actions by creating multiple $operator_{opponent}$ hierarchies. Thus, if there are two opponents, a lead and a wingman, then two separate hierarchies, specifically $operator_{lead}$ and $operator_{wingman}$ hierarchies, may be used to track their actions. All of these would be modeled within a single WCPS, so as to allow sharing among $state_{own}$ and $state_{lead}$, as well as $state_{own}$ and $state_{wingman}$, and even among $state_{lead}$ and $state_{wingman}$. In addition, AP may track their coordinated maneuvers using operators such as the $PINCER_{lead\text{-}and\text{-}wingman}$ operator introduced in the previous section. While this approach seems reasonable here, there are some issues that remain unresolved in general multiple opponent situations. For instance, suppose the two opponents attacking AP are not coordinating their actions, and they are not acting as the lead and the wingman. Should AP still use a single WCPS to track both opponents' actions? Or should it use two, one to model its interactions with one of the opponents and track that opponent's actions, and a separate one for the other opponent? Addressing these issues is a topic for future work.

The second assumption was that AP and its opponent are identical in terms of their problem spaces and operators. This assumption enabled AP to use a copy of its own problem spaces to track its opponent. This assumption is a reasonable one to make in the air-combat simulation environment, given the similarity among agents' missions, tactics and maneuvers. Furthermore, if AP does have some additional knowledge about how some of the opponent's operators differ from its own, then AP could use those operators in modeling the opponent, instead of using copies of its own operators. However, the more important point underlying this assumption is that AP has full knowledge of its opponent's overall set of problem spaces and operators. If the opponent does execute operators from outside this set, e.g., new maneuvers that AP is not familiar with, AP may not successfully track them. This is similar to the assumption in the plan recognition literature: the agent that is recognizing a plan is assumed to have full knowledge of all of the plans that the planning agent can execute [Kautz and Allen 86]. We return to this assumption in Section 5.

The third assumption was that AP can generate an accurate $state_{opponent}$. While at first this seems like a highly problematical assumption, there are several ways in which this problem is simplified. First, as mentioned earlier, a WCPS helps avoid the complex computations required for generating $state_{opponent}$. With a WCPS, various aspects of $state_{opponent}$ such as, target-aspect, angle-off, range and so on are all automatically calculated. Second, some aspects of the opponent's state are too detailed and AP can safely not model them in $state_{opponent}$. For instance, each opponent typically has a call sign, e.g., condor101, which it uses to communicate with its partners. AP does not need to model this, since it is not modeling the opponent's communications. Third, portions of $state_{opponent}$ may be generated via background information. In the air-combat simulation environment, this information includes the "intelligence" reports that are available to AP. These reports may provide the necessary information regarding the opponent's aircraft, radars, missiles, etc. and can aid AP in generating the necessary portions of $state_{opponent}$ during simulated air-combat. However, this information can be approximate or incomplete, and hence the generation of $state_{opponent}$ is not always straightforward. For instance, AP typically has background information regarding its opponent's maximum radar range — suppose the information is that the range is a maximum of 50 miles. However, given this information, AP should not directly

assume that its aircraft would be visible to the opponent's radar when AP is at a range of 50 miles from the opponent. In particular, the opponent's radar may actually have a range of only 43 miles. Furthermore, the radar may not even be pointed in the direction of AP. Thus, if AP assumes that it becomes visible to the opponent's radar as soon as the 50 miles range is reached, AP may unnecessarily give up opportunities for gaining positional advantage. In contrast, as AP moves closer and closer to the opponent, the chances of it becoming visible to the opponent's radar continually increase, and AP can commit a serious mistake if it continues to assume that the opponent cannot see it on radar. Thus, the important question that comes up here is: at what point during the combat should AP assume that it is visible on the opponent's radar? Injecting assumptions of this form into $state_{opponent}$ based on such parametric background information is not straightforward.

The solution we are experimenting with here is to inject an assumption into $state_{opponent}$ at the point where the opponent indicates a likely change in her state. One important form of this indication is a match failure. As mentioned in Section 4.2, a match failure indicates an inaccuracy in tracking the opponent's actions, and a possible explanation for this inaccuracy is a change in her actual state. In particular, if there is a change in her actual state, then $state_{opponent}$ may no longer accurately model her state, and therefore, $operator_{opponent}$ may not accurately track the opponent's actual action. Hence, a match failure is used as a trigger for injecting the assumption into $state_{opponent}$, with the expectation that it will enable $state_{opponent}$ to continue to accurately model the opponent's state. The injected assumption is then verified by corroborating the resulting $operator_{opponent}$ with the opponent's actual actions. In the above example, when AP is at the range of 50 miles from its opponent, AP "queues" the assumption that it is visible on the opponent's radar, i.e., AP gets the assumption ready for injection into $state_{opponent}$. If there is a match failure, say at the range of 42 miles, then AP injects the queued assumption into $state_{opponent}$. The resulting $operator_{opponent}$ indicates that the opponent is likely to turn to collision course. If the opponent does indeed turn to collision course, the assumption is considered corroborated, else the assumption may be withdrawn. Thus, while Section 4.2 pointed out the possibility of a full backtrack search in case of a match failure, this section suggests the introduction of an assumption(s) into $state_{opponent}$. The following section clarifies this apparent discrepancy.

## 4.5. Summary of Event Tracking

The proposed approach for event tracking is based on the following key ideas:

- A uniform mechanism for generating own flexible and reactive behaviors, and for tracking other agent's behaviors.
- A single operator hierarchy, i.e., a single interpretation for the other agent's actions, with active and passive ambiguity resolution strategies, and backtrack search.
- Use of approximation filters to avoid detailed tracking overheads.
- A world-centered representation (WCPS) for modeling continuous agent interactions.
- Assumption injection for modeling the other agent's state.

AP's approach to event tracking is presented in a high-level algorithmic form in Figure 5. The algorithm focuses on tracking a single opponent's actions, although in addition, AP also executes $operator_{own}$ hierarchy for its own actions, as well as other possible hierarchies for tracking other opponents' actions. The algorithm is based on the ideas mentioned above. It tracks the opponent's actions as presented in Section 3.2. In case of a match failure, there is an attempt to bias the backtrack search by first introducing a change in $state_{opponent}$. This bias is intended to reduce the amount of backtracking on

match failures, by first attempting to search with a changed $state_{opponent}$. Nonetheless, a full backtrack search continues to be a part of this approach (both in steps 4-c and 5), and the overheads of this search can potentially be problematical for real-time performance. Addressing this backtrack search is also a topic for future work.

1. Create an initial $state_{opponent}$ using WCPS.
2. Create an $operator_{opponent}$ hierarchy based on the existing $state_{opponent}$, while using ambiguity resolution strategies and approximation filters.
3. As long as the $operator_{opponent}$ hierarchy leads to a match success, continue with appropriate modifications in $state_{opponent}$ and the $operator_{opponent}$ hierarchy, again while continuing to use the ambiguity resolution strategies and approximations.
4. If there is a match failure, and there is a *queued* assumption(s), then:
   a. Inject all assumption(s) into $state_{opponent}$.
   b. If changed $state_{opponent}$ leads to new $operator_{opponent}$ hierarchy and a match success, then the assumption is corroborated, continue with step 3.
   c. If changed $state_{opponent}$ leads to a match failure, try selecting other operators from the $operator_{opponent}$ hierarchy. This leads to a backtrack search over the hierarchy. If this search succeeds in selecting an operator that matches the opponent's actions, then go to step 3. If this search fails, then delete the assumption, go to step 5.
5. If there is a match failure, and there is no *queued* assumption(s), then try selecting other operators from the $operator_{opponent}$ hierarchy. This leads to a backtrack search over the hierarchy. If this search succeeds in selecting an operator that matches the opponent's actions, then go to step 3. If this search fails, there is a failure in event tracking.

**Figure 5:** A high level algorithmic description of AP's event tracking

## 5. Implementation and Evaluation

An important test for the event tracking approach introduced in this article is its actual application in constructing automated pilots for simulated air-combat. Ideally, such a pilot could be constructed simply by "plugging in" the event tracking mechanism into the TacAir-Soar system. However, the underlying Soar architecture does not directly support the construction of a WCPS containing multiple operator hierarchies.[1]

We have therefore implemented an experimental variant of TacAir-Soar that can support such hierarchies. To create this variant, we started with the operators that are used by the original TacAir-Soar system. Based on our current assumption regarding similarity of pilots' behaviors, we then generated a copy of these operators to model the opponent. We then added a simple interpreter layer to Soar to enable it to support a WCPS containing multiple operator hierarchies (the interpreter itself is constructed in terms of Soar operators). The resulting TacAir-Soar implementation contains about 900 Soar rules (a single operator consists of multiple rules), and it tracks the opponent's actions as outlined in Section 4.5. Currently, this implementation focuses on tracking events in single-opponent situations. Lessons learned

[1]There is on-going research on determining whether/how the Soar architecture can support a WCPS and this may impact the final implementation of event tracking in TacAir-Soar [Jones et al. 94].

from this implementation are continuously transferred back into the original TacAir-Soar, including the sharing of representations of objects as specified in a WCPS (but not the use of multiple operator hierarchies), and the use of a single interpretation for the opponent's actions and so on.

How do we evaluate the effectiveness of this implementation for event tracking? There are at least two aspects to this question. The first question is whether the current approach enables AP, the TacAir-Soar pilot agent, to track opponent's actions accurately in real-time. The second question is whether the current approach improves AP's overall performance. This question is more complex and arises due to the fact that AP is an integrated agent with event tracking as only of its many capabilities. There is thus a need to understand the effect of the current approach on AP's overall performance. For instance, if AP spends most of its time in event tracking, that would not necessarily lead to an improvement in its performance. A detailed answer to both these concerns will have to involve extensive experiments with AP flying against human pilots in air-combat simulations. These experiments are currently planned for sometime early next year. In the meanwhile, we have attempted simpler experiments with AP flying against its clone in air-combat simulation scenarios. The clone has all of AP's operators, although it pilots an "enemy" aircraft, with different radar and missile capabilities. The advantage of this experimental set up is that it is possible to examine the operators being executed at each point by each pilot. This helps in addressing our first concern above of assessing the accuracy of event tracking. Table I shows the results of one such experiment. The experiment involves five different scenarios. All five are based on a skeletal scenario outlined by our domain experts. They differ in the the initial positions of the two aircraft involved in the combat and the missile and radar capabilities of the aircraft.

| Scen. num. | Total num. of operator executions | Percent operator executions in event tracking | Percent event track operators causing match failures | Accuracy of event tracking? |
|---|---|---|---|---|
| 1 | 37 | 8% | 0% | Yes |
| 2 | 62 | 45% | 7% | Yes |
| 3 | 101 | 45% | 28% | Yes |
| 4 | 133 | 45% | 17% | Yes |
| 5 | 138 | 48% | 20% | Yes |

**Table I:** Results of experiments in event tracking.

In Table I, the first column lists the scenario number. Each scenario takes about the same execution time, about five minutes. However, the scenarios do differ in their complexity, in terms of the maneuvers that AP and its opponent execute in the scenario. The second column in the table attempts to capture this difference. It indicates the total number of AP's operator executions in each scenario. This includes operators for AP's own actions, as well as for tracking the opponent's actions. The total number provides some indication of the comparative complexity of the different scenarios. Obviously, given that the scenarios take about the same execution time, these operators are not all applied one after another. Instead, as described in Section 3.2, AP often has to wait for the situation to change so as to apply a new operator. For instance, if AP applies the GET-MISSILE-LAR$_{own}$ operator, it may not be able to apply any other operator until it achieves its missile firing position.

The third column shows the percentage of operator executions involved in event tracking. This percentage clearly depends on the number of maneuvers that the opponent executes. The key point here is that event tracking is a non-trivial task for AP — up to 48% of the operator executions may be dedicated to event tracking. Obviously, given that AP spends much of its time waiting, this does not

translate into 48% of the total time that AP pilots its aircraft in a given simulation scenario. The fourth column shows the percentage of event tracking operators involved in match failures, including both low- and high-level operators. While some of these failures appear avoidable, some others are unavoidable — they occur because the opponent first engages in a particular maneuver and then, due to changes in its state, engages in a different maneuver. Nonetheless, the overall percentage of these operator is quite low; a maximum of only 28% of the event tracking operators are involved in match failures.

The fifth and final column indicates the accuracy of event tracking. Accuracy is checked as follows: if the opponent performs a particular maneuver, does AP track that maneuver in real-time so as to react appropriately? The answer in this column indicates that even though AP encounters match failures, it is able to backtrack or inject appropriate assumptions into $state_{opponent}$ so as to track the on-going events in real-time and respond appropriately. Thus, our basic approach to event tracking and its refinements, such as the fuzz-boxes and ambiguity resolution strategies, are able to achieve accuracy in tracking the opponent's actions and behaviors.

Clearly, this accuracy also depends on the fact that the opponent does not execute any new, unknown maneuvers. To test AP's response in situations where the opponent actually does engage in unknown maneuvers, we ran a new set of scenarios. Here, instead of adding new maneuvers to the opponent's set of maneuvers, we deleted some of AP's operators for tracking the opponent's maneuvers. Thus one or more of the opponent's maneuvers may appear like unknown maneuvers to AP. Running experiments involving these "unknown" maneuvers has illustrated some of the weaknesses in our current event tracking approach, specifically in its backtracking search. Essentially, if the opponent engages in an unknown maneuver then AP engages in backtrack search ruling out all of the possible operators. Subsequently, it begins a new effort to track the opponent's actions, but it keeps failing as long as the opponent continues its execution of the new maneuver. This repeated effort can be computationally expensive. In addition, even if the opponent subsequently engages in a known maneuver, AP may not recover in its event tracking, especially if $state_{opponent}$ is left inconsistent. Dealing with such novel maneuvers is a topic for future work.

The second question above was related to understanding the effect of the current approach in improving AP's overall performance. It is difficult to evaluate the overall performance of a complex intelligent agent like AP [Hanks et al. 93]. Obtaining quantitative estimates of the contribution of one of AP's component capabilities (such as event tracking) can prove to be even more difficult. Nonetheless, we can at least understand some of the types of benefits that AP accrues from its event tracking capability. These types are enumerated below:

1. *Event tracking is crucial for AP's survival:* It is based on event tracking that AP can recognize an opponent's missile firing behavior. Event tracking also enables AP to recognize if it managed to fire its missile before or after the opponent fired at it, via the $FIRED\text{-}FIRST_{own\text{-}and\text{-}opponent}$, mentioned in Section 4.3.

2. *Event tracking enables AP to act and react in a more intelligent manner:* Event tracking improves AP's overall understanding of a situation, and thus it can plan, act and react in a more intelligent manner. A simple example is that if the opponent is understood to be running away, AP can chase it down. However, such a chase is inappropriate if the opponent is not really running away. Similarly, if AP recognizes that the opponent is about to fire a missile, it can be more tolerant of small errors in its own missile firing position — it may not strive for an optimal positioning. These types of examples can also be seen in situations involving more than one opponent. For instance, suppose AP is facing two opponents, and that it tracks one of them as engaging in a missile evasion maneuver. It can then predict that that opponent is going to disappear from its radar, and that that opponent is

also not likely to be firing a missile while evading a missile. Based on this understanding, AP may now concentrate its attack on the second opponent. The situation would be very different if the first opponent had disappeared from its radar without engaging in the missile evasion maneuver. Without an understanding of this situation, this type of switching of AP's attack may not be possible.

3. *Event tracking helps in providing a better explanation:* An explanation capability, for explaining its own decisions to human experts, is one of AP's integral capabilities [Johnson 94]. This capability is used by the domain experts to understand AP's decision making process. If AP is seen to perform its task without tracking events appropriately, domain experts will not have sufficient confidence in its capabilities to actually use it for training or tactics development.

## 6. Summary

This article makes two contributions. First, it presents a detailed analysis of event tracking in the "real-world", dynamic, multi-agent environment of air-combat simulation. This analysis raises three novel issues for event tracking: (i) tracking agents' flexible and reactive behaviors; (ii) tracking events in the context of continuous agent interactions; (iii) tracking events in real-time. The second contribution is the new approach for event tracking. Some of the key ideas in this approach include: (i) use of a uniform mechanism for generating flexible and reactive behaviors as well as tracking them; (ii) a single interpretation for the other agent's actions with a mechanism for backtracking; (iii) active and passive ambiguity resolution strategies and approximation filters; (iv) triggered assumption injection; and (v) world-centered representation (WCPS). While this approach was introduced in the context of air-combat simulation, we expect it to generalize to other competitive or collaborative real-time, dynamic multi-agent environments, including music synthesis (for intelligent accompaniment), game playing, entertainment [Bates et al. 92], and education [Ward 91].

The article also outlined several unresolved issues. One important issue is robustness in the face of unknown events. As mentioned earlier, the system needs to improve its tracking capability in such situations. Towards that ends we are investigating an approach where AP attempts to track an opponent's novel maneuver by "self examination". Informally, this technique may be described as follows. Suppose AP observes the opponent performing a novel maneuver, say a 30 degree turn to the left, that it cannot track. It then poses itself the following question: What $\text{operator}_{\text{own}}$ would cause it (AP) to turn 30 degrees to the left? Suppose the answer is $\textit{CUT-TO-LATERAL-SEPARATION}_{\text{own}}$, then this is what the opponent is hypothesized as performing at this point. A preliminary implementation of this technique reveals that although it enables AP to track novel maneuvers, the real-time pressure makes timely self-examination difficult in some cases. Further investigations on this topic are underway. A second related issue for future work is that of reducing the backtracking overheads, particularly in situations involving such unknown events.

A third issue for future work is the automatic discovery of the size of the fuzz-box approximation for each maneuver. Experiments reveal that small reductions in the size of the current fuzz-boxes do not affect the accuracy of event tracking; although, not surprisingly, big reductions lead to possibly fatal errors in tracking. Can AP begin with large fuzz-boxes and possibly narrow them down with "experience" in simulated air-combat? A fourth important issue is modeling what AP believes the other agent knows about itself (i.e., AP). For instance, if AP believes that the other agent believes that its (AP's) missile range is larger than its actual range, AP may be able to take advantage of this situation. In particular, AP may deceive the opponent into believing that it has fired a missile without firing one. Alternatively, as the

opponent attempts to deceive AP, in some situations, AP may discover this deception. This topic is related to issues of recursive agent modeling [Gmytrasiewicz et al. 91, Wilks and Ballim 87]. Work on this topic is already underway.

A fifth issue for future work is extending event tracking to situations involving different groups of agents. Key questions that come up here include: how does a WCPS generalize to this situation? Should an agent use a single WCPS or multiple ones, particularly if the other agents are not interacting amongst themselves? Should an agent spend equal amount of time tracking each of the agents in the group, or should it be selective to avoid overwhelming itself with event tracking? Closely related to this is also the issue of abstracting away from tracking individual agents' behaviors to tracking group behaviors.

## Acknowledgement

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). Critical support has been provided by Dennis McBride of ARPA/ASTO, and Tom Brandt, Bob Richards, and Ed Harvey of BMH Inc.

## References

[Allen 83] Allen, J.
Maintaining knowledge about temporal intervals.
*Communications of the ACM* 26(11), November, 1983.

[Anderson, et al. 90]
Anderson, J. R., Boyle, C. F., Corbett, A. T., and Lewis, M. W.
Cognitive modeling and intelligent tutoring.
*Artificial Intelligence* 42:7-49, 1990.

[Azarewicz, et al. 86]
Azarewicz, J., Fala, G., Fink, R., and Heithecker, C.
Plan recognition for airborne tactical decision making.
In *National Conference on Artificial Intelligence*, pages 805-811. 1986.

[Ballard 89] Ballard, D.
Reference frames for animate vision.
In *Proceedings of International Joint Conference on Artificial Intelligence*. 1989.

[Bates et al. 92] Bates, J., Loyall, A. B., and Reilly, W. S.
*Integrating reactivity, goals and emotions in a broad agent.*
Technical Report CMU-CS-92-142, School of Computer Science, Carnegie Mellon University, May, 1992.

[Calder et al. 93] Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., Ceranowicz, A. Z.
ModSAF behavior simulation and control.
In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. March, 1993.

[Carberry 90a] Carberry, S.
Incorporating default inferences into Plan Recognition.
In *Proceedings of National Conference on Artificial Intelligence*, pages 471-478. 1990.

[Carberry 90b] Carberry, S.
*Plan Recognition in Natural Language Dialogue.*
MIT Press, Cambridge, MA, 1990.

[Decker and Lesser 93]
Decker, K., and Lesser, V.
Quantitative modeling of complex computational task environments.
In *Proceedings of the National Conference on Artificial Intelligenence.* 1993.

[Dousson et al. 93]
Dousson, C., Gaborit, P., and Ghallab, M.
Situation Recognition: Representation and Algorithms.
In *International Joint Conference on Artificial Intelligence*, pages 166-172. 1993.

[Durfee and Lesser 88]
Durfee, E. H., and Lesser, V. R.
Using Partial Global Plans to Coordinate Distributed Problem Solvers.
*Readings in Distributed Artificial Intelligence.*
Morgan Kaufmann Publishers, Palo Alto, CA, 1988.

[Gmytrasiewicz et al. 91]
Gmytrasiewicz, P. J., Durfee, E. H., and Wehe, D. K.
A decision theoretic approach to co-ordinating multi-agent interactions.
In *Proceedings of International Joint Conference on Artificial Intelligence.* 1991.

[Hanks et al. 93] Hanks, S., Pollack, M. E., and Cohen, P. R.
Benchmarks, test beds, controlled experimentation, and the design of agent architectures.
*AI Magazine* 14, 1993.

[Hill and Johnson 93]
Hill, R., and Johnson, W. L.
Impasse-driven tutoring for reactive skill acquisition.
In *Proceedings of the Conference on Intelligent Computer-aided Training and Virtual Environment Technology.* 1993.

[Johnson 94] Johnson, W. Lewis.
Agents That Learn to Explain Themselves.
In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94).* Seattle, WA, August, 1994.

[Jones et al. 93] Jones, R. M., Tambe, M., Laird, J. E., and Rosenbloom, P.
Intelligent automated agents for flight training simulators.
In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation.* March, 1993.

[Jones et al. 94] Jones, R., Laird, J. E., Tambe, M., and Rosenbloom, P. S.
Generating behavior in response to interacting goals.
In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation.* May, 1994.

[Katz 93] Katz, A.
Intelligent player -- first principle foundations.
In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation.* May, 1993.

[Kautz and Allen 86]
Kautz, A., and Allen J. F.
Generalized plan recognition.
In *National Conference on Artificial Intelligence*, pages 32-37. 1986.

[Laird and Rosenbloom 90]
Laird, J.E. and Rosenbloom, P.S.
Integrating execution, planning, and learning in Soar for external environments.
In *Proceedings of the National Conference on Artificial Intelligence*. July, 1990.

[Laird, Newell, and Rosenbloom 87]
Laird, J. E., Newell, A. and Rosenbloom, P. S.
Soar: An architecture for general intelligence.
*Artificial Intelligence* 33(1):1-64, 1987.

[McDermott and Davis 84]
McDermott, D. and Davis, E.
Planning routes through uncertain territory.
*Artificial Intelligence* 22:107-156, 1984.

[Rosenbloom, et al. 91]
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):289-325, 1991.

[Schaper et al. 94] Schaper, G. A., Pandari, S., and Singh, M.
Lookahead limits for intelligent player.
In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*. May, 1994.

[Shapiro and Rapaport 91]
Shapiro, S. C., and Rapaport, W. J.
Models and minds: knowledge representation for natural language competence.
*Philosophy and AI: essays at the interface.*
MIT Press, Cambridge, MA, 1991.

[Song and Cohen 91]
Song, F. and Cohen, R.
Temporal reasoning during plan recognition.
In *National Conference on Artificial Intelligence*, pages 247-252. 1991.

[Tambe et al. 94] Tambe, M., Jones, R., Laird, J. E., Rosenbloom, P. S., and Schwamb, K.
Building Believable Agents for Simulation Environments.
In *Proceedings of the AAAI Spring Symposium on Believable Agents*. 1994.

[Van Beek and Cohen 93]
Van Beek, P., and Cohen, R.
Resolving Plan Ambiguity for Cooperative Response Generation.
In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 938-944. 1993.

[Ward 91] Ward, B.
*ET-Soar: Toward an ITS for Theory-Based Representations.*
PhD thesis, School of Computer Science, Carnegie Mellon University, May, 1991.

[Wilks and Ballim 87]
Wilks, Y., and Ballim, A.
Multiple agents and hueristic ascription of belief.
In *Proceedings of International Joint Conference on Artificial Intelligence*. 1987.