From: Proceedings of the First International Conference on Multiagent Systems. Copyright © 1995, AAAI (www.aaai.org). All rights reserved.

Recursive Agent and Agent-group Tracking in a Real-time, Dynamic Environment

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

Abstract

Agent tracking is an important capability an intelligent agent requires for interacting with other agents. It involves monitoring the observable actions of other agents as well as inferring their unobserved actions or high-level goals and behaviors. This paper focuses on a key challenge for agent tracking: recursive tracking of individuals or groups of agents. The paper first introduces an approach for tracking recursive agent models. To tame the resultant growth in the tracking effort and aid real-time performance, the paper then presents *model sharing*, an optimization that involves sharing the effort of tracking multiple models. Such shared models are dynamically unshared as needed — in effect, a model is selectively tracked if it is dissimilar enough to require unsharing. The paper also discusses the application of recursive modeling in service of deception, and the impact of sensor imperfections. This investigation is based on our on-going effort to build intelligent pilot agents for a real-world synthetic air-combat environment.[1]

1 Introduction

In dynamic, multi-agent environments, an intelligent agent often needs to interact with other agents to achieve its goals. *Agent tracking* is an important requirement for intelligent interaction. It involves monitoring other agents' observable actions as well as inferring their unobserved actions or high-level goals, plans and behaviors.

Agent tracking is closely related to plan recognition(Kautz & Allen 1986; Azarewicz *et al.* 1986), which involves recognizing agents' plans based on observations of their actions. The key difference is that plan-recognition efforts typically focus on tracking a narrower (plan-based) class of agent behaviors, as seen in static, single-agent domains. Agent tracking, in contrast, can involve tracking a broader mix of goal-driven and reactive behaviors(Tambe & Rosenbloom 1995). This capability is important for dynamic environments where agents do not rigidly follow plans.

This paper focuses on the issues of recursive agent and agent-group tracking. Our investigation is based on an on-going effort to build intelligent pilot agents for simulated air-combat(Tambe *et al.* 1995). These pilot agents execute missions in a simulation environment called ModSAF, that is being commercially developed for the military(Calder *et al.* 1993). ModSAF provides a synthetic yet real-world setting for studying a broad range of challenging issues in agent tracking. By investigating agents that are successful at agent tracking in this environment, we hope to extract some general lessons that could conceivably be applied in other synthetic or robotic multi-agent environments(Kuniyoshi *et al.* 1994; Bates, Loyall, & Reilly 1992).

For an illustrative example of agent tracking in the air-combat simulation environment, consider the scenario in Figure 1. The pilot agent **L** in the light-shaded aircraft is engaged in combat with pilot agents **D** and **E** in the dark-shaded aircraft. Since the aircraft are far apart, **L** can only see its opponents' actions on radar (and vice versa). In Figure 1-a, **L** observes its opponents turning their aircraft in a coordinated fashion to a collision course heading (i.e., with this heading, they will collide with **L** at the point shown by **x**). Since the collision course maneuver is often used to approach one's opponent, **L** infers that its opponents are aware of its (**L**'s) presence, and are trying to get closer to fire their missiles. However, **L** has a missile with a longer range, so **L** reaches its missile range first. **L** then turns its aircraft to point straight at **D**'s aircraft and fires a radar-guided missile at **D** (Figure 1-b). Subsequently, **L** executes a 35° *fpole* turn away from **D**'s aircraft (Figure 1-c), to provide radar guidance to its missile, while slowing its rate of approach to enemy aircraft.

While neither **D** nor **E** can observe this missile on their radar, they do observe **L**'s pointing turn followed

[1] I thank Paul Rosenbloom and Ben Smith for detailed feedback on this effort. Thanks also to Lewis Johnson, Piotr Gmytrasiewicz and the anonymous reviewers for helpful comments. 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 of the Advanced Research Projects Agency and the Naval Research Laboratory.

From: Proceedings of the First International Conference on Multiagent Systems. Copyright © 1995, AAAI (www.aaai.org). All rights reserved.

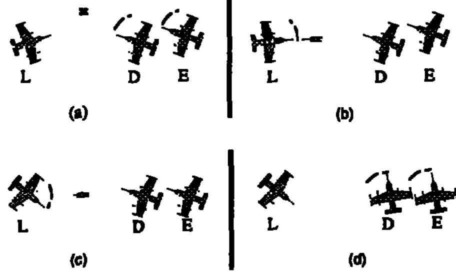

Figure 1: Pilot agents **D** and **E** are attacking **L**. An arc on an aircraft's nose shows its turn direction.

by its fpole turn. They track these to be part of L's missile firing behavior, and infer a missile firing. Therefore, they attempt to evade this missile by executing a 90° *beam* turn (Figure 1-d). This causes their aircraft to become invisible to L's radar. Deprived of radar guidance, L's missile is rendered harmless. Meanwhile, L tracks its opponents' coordinated beam turn in Figure 1-d, and prepares counter-measures in anticipation of the likely loss of its missile and radar contact.

Thus, the pilot agents need to continually engage in agent tracking. They need to track their opponents' actions, such as turns, and infer unobserved actions and high level goals and behaviors, such as the fpole, beam or missile firing behaviors. This paper focuses on two key issues in agent tracking in this environment:

- *Recursive agent tracking:* Pilot agents continually influence each other's behaviors, creating a need for recursive tracking. For instance, in Figure 1-d, to successfully track D's beam, L must also recursively track how D is likely to be tracking L's own actions — that D is aware of L's missile firing, and it is beaming in response. Such recursive tracking may also be used in service of deception, and in addressing other agents' realistic sensor (radar) limitations.
- *Agent group tracking*: An agent may need to track coordinated (or uncoordinated) activities of a group of agents, e.g., as just seen, L needed to track two coordinated opponents.

To address these issues, this paper first presents an approach for recursive tracking of an individual or a groups of agents. This approach builds upon RESC, a technique for real-time tracking of flexible and reactive behaviors of individual agents in dynamic environments. RESC is a real-time, reactive version of the *model tracing* technique used in intelligent tutoring systems — it involves executing a model of the tracked agent, and matching predictions with actual observations(Anderson *et al.* 1990; Ward 1991; Hill & Johnson 1994).

Unfortunately, recursive agent-group tracking leads to a large growth in the number of models. Executing all of these models would be in general highly problematic. The problem is particularly severe for a pilot agent, given that it has to track opponents' maneuvers and counter them in real-time, e.g., by going beam to evade a missile fired at it. Thus, for executing recursive models (and for a practical investigation of recursive tracking), optimizations for real-time performance are critical. Previous work on optimizations for agent tracking has mostly focused on *intra-model* (within a single model) optimizations, e.g., heuristic pruning of irrelevant operators(Ward 1991) restricted backtrack search(Tambe & Rosenbloom 1995), and abstraction(Hill & Johnson 1994). In contrast, this paper proposes *inter-model* (across multiple models) optimizations. It introduces an inter-model optimization called *model sharing*, which involves sharing the effort of tracking multiple models. Shared models are dynamically unshared when required. In essence, a model is selectively tracked if it is dissimilar enough to warrant unsharing. The paper subsequently discusses the application of recursive models in service of deception. This analysis is followed up with some supportive experiments.

The descriptions in this paper assume the perspective of the automated pilot agent L, as it tracks its opponents. They also assume *ideal* sensor conditions, where agents can perfectly sense each others' maneuvers, unless otherwise mentioned. Furthermore, the descriptions are provided in concrete terms using implementations of a pilot agent in a system called TacAir-Soar(Tambe *et al.* 1995), built using the Soar architecture(Newell 1990; Rosenbloom *et al.* 1991). We assume some familiarity with Soar's problem-solving model, which involves applying operators to states to reach a desired state.

2 Recursive Agent Tracking

One key idea in RESC is the uniform treatment of an agent's generation of its own behavior and tracking of other agent's behaviors. As a result, the combination of architectural features that enable an agent to generate flexible goal-driven and reactive behaviors are reused for tracking others' flexible and reactive behaviors. This uniformity is extended in this section in service of recursive agent tracking.

To illustrate this idea, we first describe L's generation of its own behaviors, using the situation in Figure 1-d, just before the agents lose radar contact with each other. Figure 2-a illustrates L's operator hierarchy when executing its fpole. Here, at the top-most level, L is executing its mission — to defend against intruders — via the *execute-mission* operator. Since the termination condition of this operator — completion of L's mission — is not yet achieved, a subgoal is generated.[2]

[2]If an operator's termination conditions remain unsatisfied, a subgoal gets created. If these termination conditions are satisfied by future state changes, then the operator and all its subgoals are terminated.

From: Proceedings of the First International Conference on Multiagent Systems. Copyright © 1995, AAAI (www.aaai.org). All rights reserved.

Different operators are available in this subgoal, such as *follow-flight-path*, *intercept*, and *run-away*. **L** selects the *intercept* operator to combat its opponent **D**. In service of *intercept*, **L** applies the *employ-missile* operator in the next subgoal. Since a missile has been fired, the *fpole* operator is selected in the next subgoal to guide the missile with radar. In the final subgoal *maintain-heading* is applied, causing **L** to maintain heading (Figure 1-d). All these operators, used for generating **L**'s own actions, will be denoted with the subscript L, e.g., $fpole_{\mathbf{L}}$. $\text{Operator}_{\mathbf{L}}$ will denote an arbitrary operator of **L**. $\text{State}_{\mathbf{L}}$ will denote the global state shared by all these operators. Together, $\text{state}_{\mathbf{L}}$ and the $\text{operator}_{\mathbf{L}}$ hierarchy constitute **L**'s model of its present dynamic self, referred to as $\text{model}_{\mathbf{L}}$.

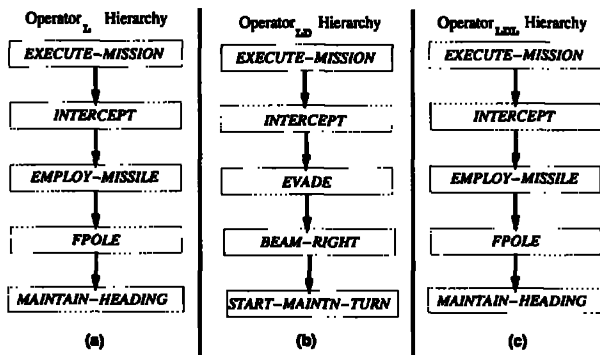

Figure 2: (a) $\text{Model}_{\mathbf{L}}$; (b) $\text{Model}_{\mathbf{LD}}$; (c) $\text{Model}_{\mathbf{LDL}}$.

$\text{Model}_{\mathbf{L}}$ supports **L**'s flexible/reactive behaviors, given Soar's architectural apparatus for operator selection and termination(Rosenbloom *et al.* 1991). **L** reuses this apparatus in tracking its opponents' behaviors. Thus, **L** uses a hierarchy such as the one in Figure 2-b to track **D**'s behaviors. Here, the hierarchy represents **L**'s model of **D**'s current operators in the situation in Figure 1-d. These operators are denoted with the subscript **LD**. This $\text{operator}_{\mathbf{LD}}$ hierarchy, and the $\text{state}_{\mathbf{LD}}$ that goes with it, constitute **L**'s model of **D** or $\text{model}_{\mathbf{LD}}$. $\text{Model}_{\mathbf{LD}}$ obviously cannot and does not directly influence **D**'s actual behavior, it only tracks **D**'s behavior. For instance, in the final subgoal, **L** applies the $\textit{start-\&-maintain-turn}_{\mathbf{LD}}$ operator, which does not cause **D** to turn. Instead, this operator predicts **D**'s action and matches the prediction with **D**'s actual action. Thus, if **D** starts turning right towards beam, then there is a match with $\text{model}_{\mathbf{LD}}$ — **L** believes that **D** is turning right to beam and evade its missile, as indicated by other higher-level operators in the $\text{operator}_{\mathbf{LD}}$ hierarchy. Note that, in reality, from **L**'s perspective, there is some ambiguity in **D**'s right turn in Figure 1-d — it could be part of a big 150° turn to run away given **L**'s missile firing. To resolve such ambiguity, **L** adopts several techniques, such as assuming the worst-case hypothesis about its enemy, which in this case is that **D** is beaming rather than running away. We will not discuss RESC's ambiguity resolution any further in this paper (see (Tambe & Rosenbloom 1995) for more details).

Thus, with the RESC approach, **L** tracks **D**'s behaviors by continuously executing the $\text{operator}_{\mathbf{LD}}$ hierarchy, and matching it against **D**'s actions. To recursively track its own actions from **D**'s perspective, **L** may apply the same technique to its recursive $\text{model}_{\mathbf{LDL}}$ (**L**'s model of **D**'s model of **L**) as shown in Figure 2-c. $\text{Model}_{\mathbf{LDL}}$ consists of an $\text{operator}_{\mathbf{LDL}}$ hierarchy and $\text{state}_{\mathbf{LDL}}$. The important point here is the uniform treatment of the $\text{operator}_{\mathbf{LDL}}$ hierarchy — on par with $\text{operator}_{\mathbf{LD}}$ and $\text{operator}_{\mathbf{L}}$ hierarchies — to support the tracking of flexible and reactive behaviors. **L** tracks $\text{model}_{\mathbf{LDL}}$ by matching predictions with its own actions. Further recursive nesting leads to the tracking of $\text{model}_{\mathbf{LDLD}}$ and so on. To track additional opponents, e.g., the second opponent **E**, **L** tracks additional models, such as $\text{model}_{\mathbf{LE}}$. **L** may also track $\text{model}_{\mathbf{LEL}}$, $\text{model}_{\mathbf{LED}}$, $\text{model}_{\mathbf{LDE}}$ etc for recursive tracking.

Recursive tracking is key to tracking other agents' behaviors in interactive situations. Thus, it is **L**'s recursive tracking of $fpole_{\mathbf{LDL}}$ which indicates a missile firing to $\text{model}_{\mathbf{LD}}$, and causes the selection of $\textit{evade-missile}_{\mathbf{LD}}$ to track **D**'s missile evasion. Note that in Figure 2-c, ambiguity resolution in $\text{model}_{\mathbf{LDL}}$ leads to an $\text{operator}_{\mathbf{LDL}}$ hierarchy that is identical to the $\text{operator}_{\mathbf{L}}$ hierarchy. One key ambiguity resolution strategy is again the worst-case assumption — given ideal sensor situations, **L** assumes **D** can accurately track **L**'s behaviors. Thus, among possible options in the $\text{operator}_{\mathbf{LDL}}$ hierarchy, the one identical to $\text{operator}_{\mathbf{L}}$ gets selected. However, these hierarchies may not always be identical and the differences between them may be exploited in service of deception at least in adversarial situations. These possibilities are discussed in more detail in Section 5.1.

3 Executing Models in Real-time

Unfortunately, the recursive tracking scheme introduced in the previous section points to an exponential growth in the number of models to be executed. In general, for N opponents, and r levels of nesting (measured with $r = 1$ for $\text{model}_{\mathbf{L}}$, $r = 2$ for $\text{model}_{\mathbf{LD}}$, and so on), the pilot agent **L** may need to execute: $\sum_{i=0}^{r-1} N^i$ models (which is r for $N = 1$, but $\frac{N^r-1}{N-1}$ for $N > 1$). This is clearly problematic given the likely scale-up in N. In particular, given its limited computational resources, **L** may be unable to execute relevant operators from all its models in real-time, jeopardizing its survival. In fact, as seen in Section 6, **L** may run into resource contention problems while executing just five models — indicating possible difficulties even for small N and r.

Thus, optimizations involving some form of selective tracking appear necessary for real-time execution of these models. Yet, such selectivity should not cause

From: Proceedings of the First International Conference on Multiagent Systems. Copyright © 1995, AAAI (www.aaai.org). All rights reserved.

an agent to be completely ignorant of critical information that a model may provide (e.g., an agent should not be ignorant of an opponent's missile firing). To this end, this paper focuses on an optimization called ***model sharing***. The overall motivation is that if there is a $\text{model}_{\mathbf{Y}}$ that is near-identical to a $\text{model}_{\mathbf{X}}$, then $\text{model}_{\mathbf{Y}}$'s states and operators can be shared with those of $\text{model}_{\mathbf{X}}$. Thus, $\text{model}_{\mathbf{Y}}$ is tracked via the execution of $\text{model}_{\mathbf{X}}$, reducing the tracking effort in half. $\text{Model}_{\mathbf{Y}}$ may be dynamically unshared from $\text{model}_{\mathbf{X}}$ if it grows significantly dissimilar. Thus, a model is selectively executed based on its dissimilarity with other models.

For an illustration of this optimization, consider $\text{model}_{\mathbf{L}}$ and $\text{model}_{\mathbf{LDL}}$, as shown in Figure 2. The $\text{operator}_{\mathbf{LDL}}$ hierarchy can be shared with the $\text{operator}_{\mathbf{L}}$ hierarchy since the two are identical. (In low-level implementation terms, sharing an $\text{operator}_{\mathbf{L}}$ involves adding a pointer indicating it is also a part of $\text{model}_{\mathbf{LDL}}$). Furthermore, information in $\text{state}_{\mathbf{LDL}}$ is shared with $\text{state}_{\mathbf{L}}$. Thus, **L** essentially executes operators from only one model, instead of two.

Given the efficiency benefits from sharing, it is often useful to abstract away from some of the differences between models in order to enable sharing. However, such abstraction may not be possible for some static and/or dynamic aspects of the models. One important aspect relates to private information. In particular, in their unshared incarnations, models have their indices organized so as to prevent a breach of privacy, e.g., $\text{model}_{\mathbf{LD}}$ can access information in $\text{model}_{\mathbf{LDL}}$, but not $\text{model}_{\mathbf{L}}$. Model sharing could potentially breach such privacy. Thus, for instance, if $\text{state}_{\mathbf{L}}$ maintains secret missile information, sharing it with $\text{state}_{\mathbf{LDL}}$ would allow $\text{model}_{\mathbf{LD}}$ to access that secret. To prevent model sharing from breaching such privacy, some aspects of the shared models may be explicitly maintained in an unshared fashion. Thus, if **L**'s missile range is (a secret) 30 miles, but **L** believes **D** believes it is 50 miles, then the missile range is maintained separately on $\text{state}_{\mathbf{L}}$ and $\text{state}_{\mathbf{LDL}}$. Figure 3 shows the resulting shared models with an unshared missile range.

Such sharing among models is related to the sharing of belief spaces in the SNePS belief representation system(Shapiro & Rapaport 1991). One key difference is ***dynamic model unsharing***. In particular, while some of aspects of the models are static (e.g., the statically unshared missile ranges above), other aspects, particularly those relating to operators, are highly dynamic. As a result, shared components may need to be dynamically unshared when dissimilar. Ideally, any two models could be merged (shared) when they are near-identical, and dynamically unshared in case of differences. This would be ideal selectivity — a model is tracked if it requires unsharing. However, in practice, both unsharing and merging may involve overheads. Thus, if an agent greedily attempts to share any two models whenever they appear near-identical, it could face very heavy overheads. Instead, it has to selectively share two models over a time period Δ so that the savings from sharing outweigh the cost of dynamic unsharing and re-merging during Δ. In particular, suppose there are two models, $\text{model}_{\mathbf{X}}$ and $\text{model}_{\mathbf{Y}}$ that are unshared over n sub-intervals $\delta_1, \delta_2, \ldots, \delta_n$ of Δ, but shared during the rest of Δ. Further, suppose $\text{cost}_\theta(\mathcal{M})$ is the cost of executing a model $\mathcal{M}$ over a time interval θ; cost(unshare) and cost(merge) are the overheads of unsharing and merging respectively; and $\text{cost}_\theta(\text{detect})$ is the cost incurred during θ of deciding if shared models need to be unshared or if unshared models can be merged (this may potentially involve comparing two different models and deciding if sharing is cost-effective). Then, in sharing $\text{model}_{\mathbf{Y}}$ with $\text{model}_{\mathbf{X}}$ during Δ, benefits outweight costs iff:

$$cost_\Delta(model_{\mathbf{Y}}) - \sum_{i=\delta_1}^{\delta_n} cost_i(model_{\mathbf{Y}}) > n \times cost(unshare) + n \times cost(merge) + cost_\Delta(detect) \quad (1)$$

While, ideally, agents may themselves evaluate this equation, our agents are unable to do so at present. Therefore, candidate categories of models — with high likelihood of sharing benefits outweighing costs — are supplied by hand. Nonetheless, agents do determine specific models within these categories that may be shared, and implement the actual sharing and dynamic unsharing. The categories are:

1. Models of distinct agents at the recursive depth of $r = 2$: If a group of agents, say **D** and **E**, together attack **L**, $\text{model}_{\mathbf{LD}}$ and $\text{model}_{\mathbf{LE}}$ may be possibly shared. Thus, if all models of its N opponents are shared, **L** may need track only one model at $r = 2$; if not shared, **L** may track N models at $r = 2$.
2. Recursive models of a single agent at $r \geq 3$: For instance, $\text{model}_{\mathbf{LDL}}$ and $\text{model}_{\mathbf{LEL}}$ may be shared with $\text{model}_{\mathbf{L}}$. Similarly, $\text{model}_{\mathbf{LDE}}$ may be shared with $\text{model}_{\mathbf{LE}}$ or $\text{model}_{\mathbf{LELE}}$, etc. Models at recursive depth $r \geq 3$ may all be shared with models at $r = 1$ or 2. If all such models are shared, **L** may need to track no models at $r \geq 3$.

The end result is that an agent **L** may track a group

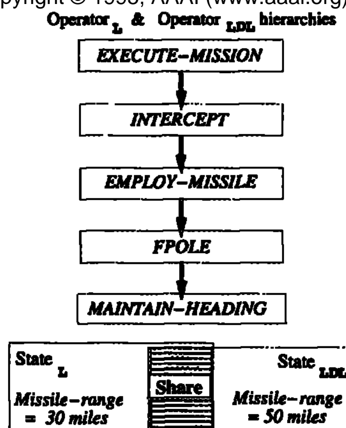

Figure 3: Sharing $\text{model}_{\mathbf{L}}$ with $\text{model}_{\mathbf{LDL}}$.

From: Proceedings of the First International Conference on Multiagent Systems. Copyright © 1995, AAAI (www.aaai.org). All rights reserved.

of N agents (with sharable models) with just two models — $model_L$ at $r = 1$, and one model at $r = 2$, and the rest are all shared — instead of $O(N^r)$ models. If the models of N agents are not sharable, then it may still need just $N + 1$ models, given the sharing in the recursion hierarchy. Thus, sharing could provide substantial benefits in tracking even for small N and r. In the following, Section 4 examines in more detail the model sharing within a group, and Section 5 examines sharing within and across recursion hierarchies.

4 Sharing in Agent-Group Tracking

Agents that are part of a single group often act in a coordinated fashion — executing similar behaviors and actions — and thus provide a possible opportunity for model sharing. For instance, if **D** and **E** are attacking **L** in a coordinated fashion, they may fly in formation, execute similar maneuvers etc. However, their actions are not perfectly identical — there are always some small delays in coordination for instance — which can be a possible hinderence in sharing. If the delays and differences among the agents' actions are small, they need to be abstracted away, to facilitate model sharing. Yet, such abstraction should allow tracking of essential group activities.

To this end, one key idea to track an agent-group is to track only a single *paradigmatic agent* within the group. Models of all other agents within the group are then shared with the model of this paradigmatic agent. Thus, a whole group is tracked by tracking a single paradigmatic agent. For example, suppose **L** determines one agent in the attacking group, say **D**, to be the paradigmatic agent. It may then only track $model_{LD}$, and share other models, such as $model_{LE}$ with $model_{LD}$, reducing its the tracking burden.

Such model sharing needs to selective, if benefits are to outweigh costs. In this case, the following domain-specific heuristics help tilt the balance in favor of sharing by reducing the cost of detection, merging and unsharing:

- Cost(detect): This involves detecting two or more agents (opponents) to be part of a group with sharable models. Such a group is detected at low cost by testing the agents' physical proximity and direction of movement. If these are within the ranges provided by domain experts, the corresponding agents' models are shared. If the agents move away from each other (outside of this range) their models are unshared. Once outside this range, no attempt is made at model sharing — such agents are likely to be engaged in dissimilar activities, and even if their models are found to be near-identical, they are likely to be so for a short time period.
- Cost(merge): Merging involves the cost of selecting a paradigmatic agent within the group. It may be possible to select an agent at random from the group for this role. However, an agent in some prominent position, such as in front of the group, is possibly a better fit for the role of a paradigmatic agent, and can also be picked out at a low cost. In air-combat simulation, an agent in such a position is typically the leader of the group of attacking opponents. It initiates maneuvers, and others follow with a small time-lag. The group leader is thus ideal as a paradigmatic agent. Note that a dynamic change in the paradigmatic agent does not cause unsharing.
- Cost(Unshare): Unsharing, however, has a rather high cost. For instance, once **D** and **E** are detected to have unshared models, a completely new $model_{LE}$ is constructed. Here, the entire $state_{LD}$ has to be copied to $state_{LE}$.

The end result is that a particular agent's model is selectively executed when the agent breaks away from the coordinated group. Otherwise, its model is merged with the paradigmatic agent's model.

5 Sharing in Recursion Hierarchy

Models of a single agent across a recursion hierarchy are likely to be near-identical to each other, and thus they form the second category of models that may allow sharing. We have so far limited our investigation of sharing/unsharing to models with $r \leq 3$, and specifically to different models of **L**, such as $model_{LDL}$ and $model_{LEL}$ at $r = 3$, with $model_L$ at $r = 1$. Other models, including those at deeper levels of nesting ($r \geq 4$) are never unshared. For instance, $model_{LDLD}$ is never unshared from $model_{LD}$. The motivation for this restriction is in part that in our interviews with domain experts, references to unshared models at $r \geq 4$ have rarely come up. In part, this also reflects the complexity of such unsharing, and it is thus an important issue to be addressed in future work.

To understand the cost-benefit tradeoffs of sharing recursive models, it is first useful to understand how sharing and unsharing may actually occur. One general technique for accomplishing sharing in the recursion hierarchy is to first let $model_L$ generate its operator hierarchy. As the hierarchy is generated, if $model_{LDL}$ agrees with an $operator_L$ — that is, it would have generated an identical $operator_{LDL}$ given $state_{LDL}$ — then it ($model_{LDL}$) "votes" in agreement. This "vote" indicates that that particular $operator_L$ from $model_L$ is now shared with $model_{LDL}$. This essentially corresponds to the worst case strategy introduced in section 2 — given a choice among $operators_{LDL}$, the one that is identical to $operator_L$ is selected and shared.

Thus, the detection/merging cost is low, since this can be accomplished without an extensive comparison of models. Furthermore, the savings from model sharing are substantial — as discussed below, unsharing occurs over small time periods. Furthermore, the unsharing cost is low, since it does not involve state copying. Thus, sharing benefits appear to easily outweigh

From: Proceedings of the First International Conference on Multiagent Systems. Copyright © 1995, AAAI (www.aaai.org). All rights reserved.

its costs.

Unsharing actually occurs because of differences between $\text{state}_{\mathbf{L}}$ and $\text{state}_{\mathbf{LDL}}$. Due to these differences, the recursive $\text{model}_{\mathbf{LDL}}$ cannot generate an operator that is shared with $\text{operator}_{\mathbf{L}}$. There is then unsharing of the operator hierarchies in $\text{model}_{\mathbf{LDL}}$ and $\text{model}_{\mathbf{L}}$, which may be harnessed in service of deceptive (or other) tactics. In the following, Subsection 5.1 focuses on one general strategy for such deception. Subsection 5.2 focuses on a special class of differences between recursive states — caused by sensor imperfections — and the deceptive maneuvers possible due to those differences.

5.1 Deception

Due to differences between $\text{state}_{\mathbf{LDL}}$ and $\text{state}_{\mathbf{L}}$, $\text{model}_{\mathbf{LDL}}$ may generate an $\text{operator}_{\mathbf{LDL}}$ that cannot be shared in the $\text{operator}_{\mathbf{L}}$ hierarchy. This indicates to **L** that **D** expects **L** to be engnaged in a different maneuver ($\text{operator}_{\mathbf{LDL}}$) than the one it is actually executing ($\text{operator}_{\mathbf{L}}$). In such cases, **L** may attempt to deceive **D** by abandoning its on-going maneuver and "playing along" with what it believes to be **D**'s expectations.

To understand this deceptive strategy, consider the following case of **L**'s deceptive missile firing. Let us go back to the situation in Figure 1-a, although now, assume that $\text{state}_{\mathbf{L}}$ maintains a secret missile range of 30 miles, while $\text{state}_{\mathbf{LDL}}$ maintains the range to be 50 miles. The missile range is noted in the unshared portions of the states as shown in Figure 3. At a range of 50 miles — given that $\text{state}_{\mathbf{LDL}}$ notes the missile range to be 50 miles — $\text{model}_{\mathbf{LDL}}$ suggests the execution of a *employ-missile*$_{\mathbf{LDL}}$ operator. This causes unsharing with operators in $\text{model}_{\mathbf{L}}$. *Employ-missile*$_{\mathbf{LDL}}$ subgoals into *get-steering-circle*$_{\mathbf{LDL}}$, indicating a turn to point at target, as shown in Figure 1-b.

These operators suggest actions for **L** in order to deceive its opponent. **L** may execute deceptive $\text{operators}_{\mathbf{L}}$ that create the external actions suggested by $\text{operator}_{\mathbf{LDL}}$ *without actually launching a missile.* **L** therefore executes a *employ-missile-deceptive*$_{\mathbf{L}}$ operator. This subgoals into the *get-steering-circle-deceptive*$_{\mathbf{L}}$ operator. This causes the next subgoal, of *start-&-maintain-turn*$_{\mathbf{L}}$ in $\text{model}_{\mathbf{L}}$ which actually causes **L** to turn to point at its target, **D**. This difference in $\text{model}_{\mathbf{L}}$ and $\text{model}_{\mathbf{LDL}}$ causes some unsharing in their operator hierarchies, as shown in Figure 4. After pointing at target, $\text{model}_{\mathbf{LDL}}$ executes the *fpole*$_{\mathbf{LDL}}$ operator — that is, **L** believes that **D** is expecting **L**'s fpole to support an actual missile in the air. **L** executes *fpole-deceptive*$_{\mathbf{L}}$ without actually firing a missile. Thus, with a deceptive maneuver, **L** convinces **D** that it has fired a missile at a much longer range, without actually firing one — forcing **D** to go on the defensive by turning towards beam.

L can employ a whole class of such deceptive maneuvers by going along with $\text{model}_{\mathbf{LDL}}$'s expectation, as it did here. This is essentially a general strategy for deceptive maneuvers, which is instantiated with particular deceptive maneuvers in real-time. Yet, this is only a first step towards a full-fledged deceptive agent. There are many other deceptive techniques and issues that remain unresolved, e.g., determining whether engaging in deception would lead to a globally sub-optimal behavior.

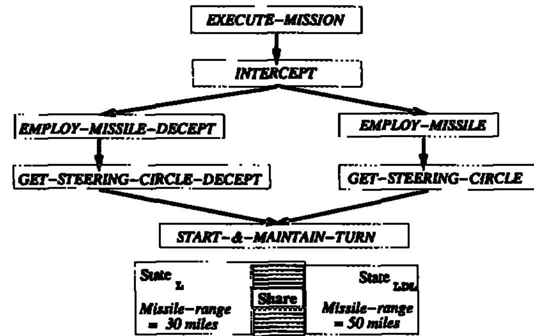

Figure 4: Deceptive missile firing: $\text{operator}_{\mathbf{L}}$ and $\text{operator}_{\mathbf{LDL}}$ hierarchies are dynamically unshared.

5.2 Sensor Imperfections

Realistic radar imperfections in this domain also lead to unsharing among recursive models. It is useful to examine these in some detail, since these are illustrative of the types of differences that are expected to arise in other domains where agents have realistic sensors. To this end, it is useful to classify the different situations resulting from these imperfections as shown in Figure 5. As a simplification, these situations describe **L**'s perspective as it interacts with a single opponent for **D**, and limited to $r \leq 3$. Figure 5-a focuses on an agent's awareness of another's presence. In the figure, Aware<BAZ> denotes someone's awareness of an agent named BAZ. Furthermore, subscript **L** indicates **L**'s own situation, a subscript **LD** indicates **L**'s tracking of **D**, a subscript **LDL** indicates **L**'s recursive tracking of **D**'s tracking of **L**. Thus, the first branch point in 5-a indicates whether **L** is aware of **D**'s presence ($+\text{Aware}_{\mathbf{L}}$<D>) or unaware ($-\text{Aware}_{\mathbf{L}}$<D>). If $-\text{Aware}_{\mathbf{L}}$<D> then **L** can not track **D**'s awareness. If $+\text{Aware}_{\mathbf{L}}$<D>, then **L** may believe that **D** is aware of **L**'s presence ($+\text{Aware}_{\mathbf{LD}}$<L>) or unaware ($-\text{Aware}_{\mathbf{LD}}$<L>). If $+\text{Aware}_{\mathbf{LD}}$<L>, then **L** may have beliefs about **D**'s beliefs about **L**'s awareness: $+\text{Aware}_{\mathbf{LDL}}$<D> or $-\text{Aware}_{\mathbf{LDL}}$<D>.

While an agent may be aware of another, it may not have accurate sensor information of the other agent's actions, specifically, turns, climbs and dives. For instance, in Figure 1-d, $+\text{Aware}_{\mathbf{L}}$<D>, yet **L** loses radar contact due to **D**'s beam. Figure 5-b classifies these situations. Here, $+\text{Sense}_{\mathbf{L}}$<D> refers

From: Proceedings of the First International Conference on Multiagent Systems. Copyright © 1995, AAAI (www.aaai.org). All rights reserved.

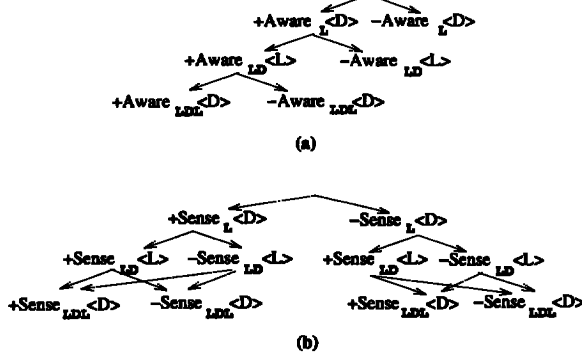

Figure 5: Classifications by: (a) awareness; (b) accuracy of sensor information.

to situations where **L** believes it has accurate sensor information of **D**'s actions, while $-\text{Sense}_{L}$<D> refers to situations as in Figure 1-d, where it does not. In either case, **L** may believe that **D** either has accurate sensor information of **L**'s actions ($+\text{Sense}_{LD}$<L>) or not ($-\text{Sense}_{LD}$<L>). Thus, in Figure 1-d, while $-\text{Sense}_{L}$<D>, **L** also believes **D** has lost radar contact due to its 90° beam turn ($-\text{Sense}_{LD}$<L>). Recursion continues with $+\text{Sense}_{LDL}$<D> and $-\text{Sense}_{LDL}$<D>.

Based on the above classification, **L**'s perspective of a situation may be described as a six-tuple. For instance, Figures 1-a to 1-c may be described as ($+\text{Aware}_{L}$<D>, $+\text{Aware}_{LD}$<L>, $+\text{Aware}_{LDL}$<D>, $+\text{Sense}_{L}$<D>, $+\text{Sense}_{LD}$<L>, $+\text{Sense}_{LDL}$<D>). This is the previously introduced *ideal* sensor situation, with "+" awareness and sensor accuracy. Based on the six-tuple, 64 such situations seem possible. However, many are ruled out — if an agent is unaware of another, it cannot have accurate sensor information regarding that agent — reducing the number of possible situations to 15.

Within these 15, we have so far examined unsharing and deception in the context of one situation, namely the ideal situation. We now briefly examine the unsharing and deception possible in the remaining 14 situations. Among these 14, there are three that typically arise in the initial portions of the combat where **L** believes **D** is unaware of **L** ($-\text{Aware}_{LD}$<L>). For instance, **L** may have seen **D** by virtue of its longer range radar, but it may have assumed that **D** is still unaware due to its shorter range radar: ($+\text{Aware}_{L}$<D>, $-\text{Aware}_{LD}$<L>, $-\text{Aware}_{LDL}$<D>, $+\text{Sense}_{L}$<D>, $-\text{Sense}_{LD}$<L>, $-\text{Sense}_{LDL}$<D>). In all these cases, model_{LDL} is null, and thus the question of sharing with model_{L} does not arise. Suppose as the aircraft move even closer, **D** engages in the collision course maneuver, which allows **L** to conclude that $+\text{Aware}_{LD}$<L>. Here, there are two possibilities. First, if $-\text{Aware}_{LDL}$<D>, i.e., **L** believes **D** believes **L** is unaware of **D**, there is much greater dissimilarity between model_{LDL} and model_{L}. Model_{LDL} now predicts that **L** will not engage in combat with **D**, i.e., there will be unsharing even with the $\textit{intercept}_{L}$ operator. Once again, **L** may deceive **D** by acting consistent with model_{LDL}'s expectation, and not turn towards **D**. This is similar to the deceptive strategy introduced in Section 5.1. **L** may then wait till **D** gets closer and then turn to attack.

The second possibility is $+\text{Aware}_{LDL}$<D>. In this case, we return to the *ideal* situation in Figures 1-a to 1-c, where unsharing is still possible as in Figure 4. Furthermore, even with $+\text{Aware}_{LDL}$<D>, there are situations with $-\text{Sense}_{LD}$<L>, where **L** believes **D** cannot sense **L**'s actions. In such cases, **L** may engage in deception by deliberately *not* acting consistent with model_{LDL}'s expectations, e.g., diving when model_{LDL} does not expect such a dive. Such deliberate unsharing is another type of deceptive strategy that among many others, is one we have not examined in detail so far.

6 Experimental Results

To understand the effectiveness of the agent tracking method introduced here, we have implemented an experimental variant of TacAir-Soar(Tambe *et al.* 1995). The original TacAir-Soar system contains about 2000 rules, and automated pilots based on it have participated in combat exercises with expert human pilots(Tambe *et al.* 1995). Our experimental version — created since it employs an experimental agent tracking technology — contains about 950 of these rules. This version can recursively track actions of individuals or groups of opponents while using the model sharing optimizations, and engaging in deception. Proven techniques from this experimental version are transferred back into the original TacAir-Soar.

Table 1 presents experimental results for typical simulated air-combat scenarios provided by the domain experts. Column 1 indicates the number of opponents (N) faced by our TacAir-Soar-based agent **L**. Column 2 indicates whether the opponents are engaged in a coordinated attack. Column 3 shows the actual maximum number of models used in the combat scenarios with the optimizations (excluding temporary model unsharing in service of deception). The numbers in parentheses are projected number of models — $2N+1$ — without the model sharing optimization (the actual number without sharing should be $O(N^r)$, but we exclude the permanently shared models from this count — see Section 5). With optimizations, as expected, the number of models is $N+1$ when opponents are not coordinated, and just two when the opponents are coordinated. Column 4 shows the actual and projected number of operator executions. The projected number is calculated assuming $2N+1$ models. Column 5 shows a two to four fold reduction (projected/actual) in the number of operators. Savings are higher with coordinated opponents. **L** is usually successful in real-time

From: Proceedings of the First International Conference on Multiagent Systems. Copyright © 1995, AAAI (www.aaai.org). All rights reserved.

tracking in that it is able to track opponents' behaviors rapidly enough to be able to respond to them. **L** is unsuccessful in real-time tracking in the case of four uncoordinated opponents (with 5 models), and it gets shot down (hence fewer total operators than the case of 2 opponents). This failure indicates that our optimizations *have helped* — without them, **L** could have failed in all cases of 2 or 4 opponents since they involve 5 or more projected models. It also indicates that **L** may need additional optimizations.

N	Coord?	Actual(projct) num max model	Actual(projct) total operatrs	Reduction (proj/act)
1	–	2 (3)	143(213)	1.5
2	No	3 (5)	176(314)	1.8
4	No	5 (9)	148(260)	1.8
2	Yes	2 (5)	109(251)	2.3
4	Yes	2(9)	105(407)	3.9

Table 1: Improvements due to model sharing.

7 Summary

This paper focused on real-time recursive tracking of agents and agent-groups in dynamic, multi-agent environments. Our investigation was based on intelligent pilot agents in a real-world synthetic air-combat environment, already used in a large-scale operational military exercise(Tambe *et al.* 1995). Possible take-away lessons from this investigation include:

- Address recursive agent tracking via a uniform treatment of the generation of flexible/reactive behaviors, as well as of tracking and recursive tracking.
- Alleviate tracking costs via model sharing — with selective unsharing in situations where models grow sufficiently dissimilar.
- Track group activities by tracking a paradigmatic agent.
- Exploit differences in an agent's self model and its recursive self model in service of deception and other actions.

One key issue for future work is understanding the broader applicability of these lessons. To this end, we plan to explore the relationships of our approach with formal methods for recursive agent modeling(Gmytrasiewicz, Durfee, & Wehe 1991; Wilks & Ballim 1987). This may help generalize the tracking approach introduced in this paper to other multi-agent environments, including ones for entertainment or education.

References

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

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

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

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

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

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

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

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

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

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

Shapiro, S. C., and Rapaport, W. J. 1991. *Models and minds: knowledge representation for natural language competence*. Cambridge, MA: MIT Press.

Tambe, M., and Rosenbloom, P. S. 1995. Resc: An approach to agent tracking in a real-time, dynamic environment. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*

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

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

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