Architectures for Agents that Track Other Agents in Multi-agent Worlds

Milind Tambe and Paul S. Rosenbloom

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

Abstract. In multi-agent environments, an intelligent agent often needs to interact with other individuals or groups of agents to achieve its goals. *Agent tracking* is one key capability required for intelligent interaction. It involves monitoring the observable actions of other agents and inferring their unobserved actions, plans, goals and behaviors. This article examines the implications of such an agent tracking capability for agent architectures. It specifically focuses on real-time and dynamic environments, where an intelligent agent is faced with the challenge of tracking the highly flexible mix of goal-driven and reactive behaviors of other agents, in real-time. The key implication is that an agent architecture needs to provide direct support for flexible and efficient reasoning about other agents' models. In this article, such support takes the form of an architectural capability to execute the other agent's models, enabling mental simulation of their behaviors. Other architectural requirements that follow include the capabilities for (pseudo-) simultaneous execution of multiple agent models, dynamic sharing and unsharing of multiple agent models and high bandwidth inter-model communication.
We have implemented an agent architecture, an experimental variant of the Soar integrated architecture, that conforms to all of these requirements. Agents based on this architecture have been implemented to execute two different tasks in a real-time, dynamic, multi-agent domain. The article presents experimental results illustrating the agents' dynamic behavior.

1 Introduction

In a multi-agent environments, intelligent agents often need to interact with each other, either collaboratively or non-collaboratively, to achieve their goals. Many of these multi-agent domains are real-time and dynamic, requiring the interaction to be highly flexible and reactive, as well as real-time. Examples of such environments include applications in arenas such as education, entertainment, and training. For instance, in the education arena, intelligent tutoring systems need to interact with students in real-time[32]. In the arena of entertainment, recent work has focused on real-time, dynamic interactivity among multiple agents within virtual reality environments[5, 12, 17]. Similarly, in the arena of training, there is a recent thrust on dynamic, real-time interactive simulations[24, 26, 33]. In these simulations, humans may interact with tens or hundreds of intelligent agents, as they participate in realistic traffic environments that simulate traffic jams and pedestrians[8], or air-traffic control environments that simulate multiple aircraft on airfields[19], or large-scale combat environments that simulate friendly and enemy troops[30]. Such real-time interaction is also seen in robotic environments, particularly in work such as robotic collaboration without communication[16].

In all these environments, *agent tracking* is a key capability required for intelligent interaction. It involves monitoring other agents' observable actions and inferring their unobserved actions or high-level goals, plans and behaviors. This capability is important in both collaborative and non-collaborative settings. Certainly, in non-collaborative settings, it is usually not in a competitor's interest to directly communicate its goals and plans to an agent — the agent thus needs to infer them to compete effectively. Even in collaborative settings, such communication may not be possible due to the cost or the risk involved, the lack of a common communication language, or inexpressivity of the common communication language; creating a need for an agent tracking capability for effective collaboration[14].

This agent tracking capability is closely related to plan recognition, which involves recognizing agents' plans based on observations of their actions[15, 2, 23]. One key difference is that plan-recognition efforts typically focus on tracking a narrower (plan-based) class of agent behaviors, as seen in static, single-agent domains. In particular, they assume that agents rigidly follow plans step-by-step. Agent tracking, in contrast, can involve tracking a broader mix of goal-driven and reactive behaviors. This capability is important for dynamic environments where agents do not rigidly follow plans.

This article discusses the implications of such an agent tracking capability for agent architectures — specifically, ways in which an architecture may facilitate agent tracking.[1] Why seek architectural support for this capability? There are at least two key reasons. First, agent tracking is a ubiquitous capability in multi-agent worlds. As discussed above, irrespective of whether agents are engaged in collaborative or competitive activity, agent tracking is essential to their interaction. It would be more efficient to provide architectural support for such a capability. Second, recent research in the fields of cognitive and developmental psychology has focused on the *theory of mind* hypothesis[4]. This hypothesis suggests that an innate (neurocognitive) capability has evolved to enable humans to ascribe mental states to others[3]. This research appears to indicate that automated intelligent agents would be well-served by an architectural capability to reason about other agents' mental states.

The article begins with an analysis (in Section 2) of some of the key requirements for agent tracking in real-time, dynamic environments. This analysis is based on tasks in a real-world, multi-agent environment and assumes that an agent is situated in the environment, as it tracks other agents while simultaneously interacting with them. Key requirements revealed by this analysis include:

1. Tracking other agents' highly flexible mix of goal-driven and reactive behaviors.
2. Recursively tracking its own actions from the perspective of other agents, so as to understand their impact on the other agents' behaviors.
3. Tracking groups of other agents, possibly acting in coordination.

[1] This article is based on our previous work on agent tracking[28, 25]. However, it focuses mainly on architectural implications of that work, rather than the tracking capability itself.

4. Simultaneously tracking and reacting to other agents' actions.
5. Tracking other agents' activities in real-time, while resolving ambiguities.

Section 3 presents an approach to agent tracking that addresses the first item above: tracking flexible and reactive behaviors. The approach is based on the *model tracing* technique used in intelligent tutoring systems (ITS) for tracking student actions[1, 32]. To track the activities of a student, an ITS executes a model of that student to generate predictions. Tracking proceeds by matching these predictions with actual observations. However, as with plan recognition, previous ITS work has primarily focused on static environments.[2] This article describes the application of model tracing in real-world dynamic tasks, where agents exhibit a complex mix of goal-driven and reactive behaviors. Section 3 illustrates that in tracking such behaviors, one key implication for agent architectures is the capability to execute models of other agents. That is, an architecture must not only generate an agent's own behaviors; it must, in addition, execute models of other agents.

Section 4 discusses the architectural implications of the next two issues from the above list: recursive agent tracking and agent-group tracking. Recursive agent tracking requires an architecture to support the execution of the agent's recursive model, i.e., an agent's model of some other agent's model of itself (the original agent). For example, to recursively track an agent **D**, its architecture needs to be capable of executing its (**D**'s) model of some other agent's model of itself (**D**). Similarly, tracking agent-groups requires the architecture to be capable of execution of an agent's models of all of the different agents in a group. Unfortunately, the recursive tracking of a large group of agents can involve the execution of a large number of models. In particular, the presence of N other agents and r levels of recursively nested models may create a need for tracking an exponential number ($O(N^r)$) of models. Since this computational cost is unacceptable, especially in real-time environments, an architecture needs to support heuristic optimizations such as *model sharing* to reduce the number of models it has to execute. Thus, an agent architecture may need the capability to share multiple similar models. Such shared models may need to be dynamically unshared when they grow dissimilar.

Section 5 discusses the fourth item on the list: simultaneous tracking and reacting. This requires an architecture to be capable of (pseudo-)simultaneous execution of multiple models, essentially to simulate the simultaneity in agents' actions in the world.

The fifth and final item on the above list is one of real-time tracking with accurate resolution of ambiguities. This article will only briefly touch upon this last item. Previous model tracing and plan recognition systems have certainly dealt with the problem of ambiguity resolution — although, many of these solutions are not necessarily intended for real-time environments[28, 27]. Elsewhere, we have presented an approach called RESC (for *REal-time Situated Commitments*), that builds upon the technique from Section 3, and aims to resolve ambiguities in real-time[28]. The architectural implications of agent tracking described in this article are not dependent on RESC. Nonetheless, we

[2] There are some recent ITS applications that have ventured into dynamic environments, e.g., REACT[13], but they still primarily rely upon a plan-driven tracking strategy, dealing with the dynamic aspects as exceptions.

briefly describe RESC in the following as an example technique for real-time ambiguity-resolution: RESC's situatedness is based on its ability to continuously track the other agents' actions in the *current* world state. Despite the ambiguities it faces, RESC quickly commits to a single interpretation of the other agents' on-going actions — there is no exhaustive examination of alternatives. These commitments constrain future tracking and interpretation. Thus, commitments in RESC play the same constraining role in tracking, as they do in constraining planning in some plan-based architectures, such as IRMA[6]. With additional information becoming available later, these commitments may turn out to be inappropriate. In such cases, the interpretations are revised in real-time, via an on-line repair mechanism. RESC's real-time character derives from its situatedness, its quick commitments, and its on-line repair.

Section 6 presents an implementation of an agent architecture, a variation on the Soar integrated architecture[18, 22], that conforms to the requirements outlined above. In our descriptions, we assume some familiarity with Soar's problem-solving model, which involves applying operators to states to reach a desired state. Section 6 also presents experimental results illustrating the dynamic behavior of agents based on this architecture. Section 7 presents related work, and Section 8 concludes.

2 Agent Tracking in a Real-world Setting

Our investigation of agent tracking is based on an on-going effort to build intelligent pilot agents for a synthetic combat environment[26]. This environment is based on a commercially developed simulator called ModSAF[7], which has already been used in an operational military exercise involving human participants. ModSAF provides a synthetic yet real-world setting for studying a broad range of challenging issues in agent tracking. Given the real-world nature of this environment, we expect that lessons learned here from analysis of agent tracking will be broadly applicable to other real-time, dynamic multi-agent environments, such as the ones discussed at the beginning of Section 1.

For an illustrative example of agent tracking in this combat simulation environment, consider first the air-to-air combat scenario in Figure 1, involving fighter jets. 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. Given a highly hostile environment, **L** may also infer that opponents are closing in 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^o *fpole* turn away from **D**'s aircraft (Figure 1-c), to provide radar guidance to its missile, while slowing its rate of approach to the enemy aircraft.

While neither **D** nor **E** can observe this missile on their radar, they do observe **L**'s pointing turn followed by its fpole turn. They track these to be part of **L**'s missile firing

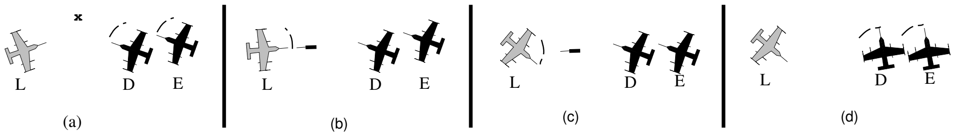

Fig. 1. Pilot agents **D** and **E** are engaged in combat with **L**. An arc on an aircraft's nose shows its turn direction.

behavior, and infer a missile firing. Therefore, they attempt to evade this missile by executing a 90^o *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, in Figure 1-d, **L** tracks its opponents' coordinated beam turn (even while not seeing the complete turn). **L** then prepares counter-measures in anticipation of the likely loss of both 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. Agent tracking in this real-time, dynamic, multi-agent environment can be seen to raise the key issues outlined in Section 1. More specifically:

1. *Tracking flexible and reactive behaviors:* Pilot agents must track other agents' highly flexible mix of goal-driven and reactive behaviors. For instance, in Figure 1, pilot agents need to track each other's continuous reactions. Indeed, if **D** and **E** had executed a 90^o *pre-emptive beam* turn prior to Figure 1-b, **L** would have needed to track that and react by not going through with its missile firing.
2. *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.
3. *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.
4. *Simultaneous acting and tracking:* As participants in their environment, pilot agents must track opponents' maneuvers and simultaneously react appropriately. For instance, **D** and **E** need to track **L**'s behaviors, as they maneuver their own aircraft.
5. *Real-time ambiguity resolution*: Pilot agents need to resolve ambiguities in other agent's actions in real-time. To survive in the scenario presented in Figure 1, **D** and **E** need to correctly interpret **L**'s missile firing in real-time, although alternative interpretations of **L**'s turns are possible.

Many of these same challenges also come up in other tasks in the combat simulation environment. For example, one such task involves a team of helicopters trying to follow a flight plan, as shown in Figure 2-a. Here, as a participant in the team activity, a subordinate may be required to track the leader (in the front) and possibly other members, to detect distinct changes in their method of flight, e.g., following the ground contour, flying nap-of the earth, or flying steady altitude[29], or to avoid collisions with teammates, especially when they are about to break formation. At a specified (holding) point (shown by **x**), the leader and other members may start hovering, an indication to all the teammates that they should wait at that point. The leader then pulls ahead, examines the forward area (Fig 2-b), and returns to its teammates to bring them forward. Here, its return may indicate to the teammates that they should get ready to start following (Fig 2-c). The leader may not be able to verbally communicate all such information, since the mission may require radio silence to avoid detection by enemy forces. It then becomes essential for an intelligent agent to infer relevant information (e.g., that the team has reached the holding area) from observation of the actions of individual team members. Thus, agent tracking in this context of helicopters also raises some of the issues outlined above such as tracking groups of agents, tracking their flexible and reactive behaviors, as well as simultaneous tracking and acting.

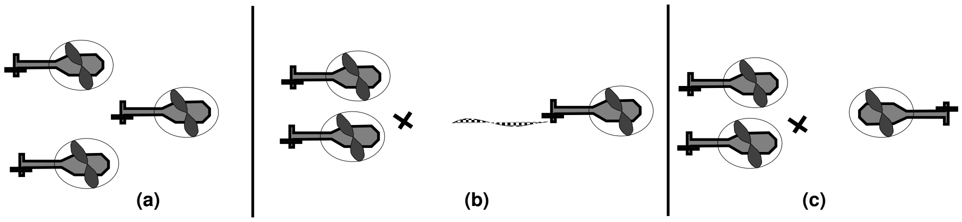

Fig. 2. A simulated combat scenario involving a team of helicopters.

3 Tracking Flexible and Reactive Behaviors

Our basic approach for tracking an agent's actions is to execute a model of that agent, while matching the model's predictions against the agent's actual actions. To track an agent's behaviors in a dynamic environment, it is necessary to be able to execute that agent's model in a flexible fashion — the execution must be responsive to the dynamic changes in the world situation. One key observation here is that the agent doing the tracking is itself a participant in the environment: it is capable of the rich flexible and reactive behaviors required in this environment. That is, its architecture can execute such behaviors in this environment. Therefore, this architecture may be reused to execute a model of another agent, to allow for flexible and reactive model execution. There is thus uniformity in an agent's generation of its own behaviors, and its tracking of other agent's behaviors.

To understand the above in concrete terms, consider the fighter-jet air-combat scenario in Figure 1. Here, **D** may track its opponent **L**'s actions, by utilizing uniformity in acting and tracking. (The following description assumes agents implemented in the Soar architecture, although the key concept of uniformity in acting and tracking is not specific to Soar.) **D**'s internal state and operator hierarchy are as depicted in Figure 3-a. **D**'s internal state maintains information regarding mission specifications, and receives input from its radar sensor. Based on the state, **D** makes appropriate operator selections to generate the desired behavior in its external environment. Here, it has selected *execute-mission* as the top operator. Since the termination condition of this operator — completion of **D**'s mission — is not yet achieved, a subgoal is generated. The *intercept* operator is selected in that subgoal. In the following subgoal, the *employ-missile* operator is selected. The subgoal after that applies *get-firing-position* to get to a missile firing position. Skipping down to the final subgoal, *maintain-heading* enables **D** to maintain heading, as seen in Figure 1-b. The operators in Figure 3-a used for generating **D**'s own actions will henceforth be denoted with the subscript **D**, e.g., $intercept_{\mathbf{D}}$. $\text{Operator}_{\mathbf{D}}$ will denote an arbitrary operator of **D**. $\text{State}_{\mathbf{D}}$ will denote the state. Together, $\text{state}_{\mathbf{D}}$ and the $\text{operator}_{\mathbf{D}}$ hierarchy may be considered as **D**'s model of its present self, referred to as $\text{model}_{\mathbf{D}}$.

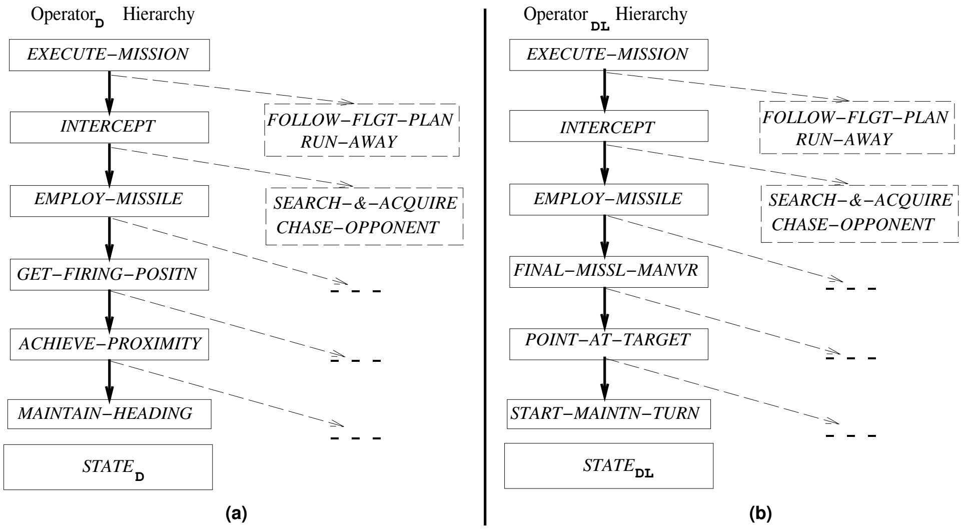

Fig. 3. (a) $\text{Model}_{\mathbf{D}}$, (b) $\text{Model}_{\mathbf{DL}}$. Solid lines indicate the actual operator hierarchy; dashed lines indicate unselected alternatives, e.g., *run-away* is an alternative to *intercept*.

$\text{Model}_{\mathbf{D}}$ supports **D**'s flexible/reactive behaviors via its embedding within Soar; and in particular, via two of Soar's architectural features: (i) a decision procedure that supports flexibility by integrating all available knowledge about preferences among candidate operators before deciding to commit to a single operator; (ii) termination conditions for operators that support reactivity by terminating operators in response to the given situation[22]. These architectural features get reused when other agents' models are executed on the same architecture. To illustrate this re-use, we assume for now that **D** and **L** possess an identical set of maneuvers.[3] Thus, **D** uses a hierarchy such

[3] This is not a necessary condition. The main requirement is for a reasonably accurate model of the other agent's possible maneuvers. Bounded deviations from this model, e.g., differences in

as the one in Figure 3-b to track **L**'s behaviors. Here, the hierarchy represents **D**'s model of **L**'s current operators in the situation of Figure 1-b. These operators are denoted with the subscript **DL**. The operator$_{\mathbf{DL}}$ hierarchy, along with the state$_{\mathbf{DL}}$ that goes with it, constitute **D**'s model of **L** or model$_{\mathbf{DL}}$. Model$_{\mathbf{DL}}$ obviously cannot influence **L**'s actual behavior. For instance, in the final subgoal **D** has applied the *start-&-maintain-turn*$_{\mathbf{DL}}$ operator to state$_{\mathbf{DL}}$. This operator cannot cause **L** to turn. It predicts **L**'s turn and matches the prediction against **L**'s actual action. Thus, if **L** starts turning to point at **D**'s aircraft, then there is a match with model$_{\mathbf{DL}}$'s predictions. Given this match, **D** now believes **L** is turning to point at its target, (i.e., **D**), to fire a missile, as indicated by other higher-level operators in the hierarchy. **D** tracks **L**'s behaviors in this manner by continuously executing the operator$_{\mathbf{DL}}$ hierarchy, and matching it with **L**'s actions.

Thus, **D**'s architecture, in addition to generating its own behaviors via execution of model$_{\mathbf{D}}$, gets reused to track other agents' behaviors via execution of models such as model$_{\mathbf{DL}}$. This provides the requisite functionality for flexible and reactive model execution. In particular, operator$_{\mathbf{D}}$ and operator$_{\mathbf{DL}}$ are selected and terminated in the same flexible manner. For instance, as state$_{\mathbf{DL}}$ changes, an operator$_{\mathbf{DL}}$ terminates if its termination conditions are satisfied, and new operators$_{\mathbf{DL}}$ get selected. Thus, execution of model$_{\mathbf{DL}}$ can be sensitive to dynamic changes in the world situation.

4 Recursive Agent and Agent-Group Tracking

The approach presented in the previous section may be extended in a straightforward manner for both recursive agent tracking and agent-group tracking. The key is to execute appropriate agent models[25]. Recursive tracking requires the execution of a recursive model. Consider, for instance, **D**'s recursive tracking in Figure 1. **D** can recursively track its own actions from **L**'s perspective, by executing model$_{\mathbf{DLD}}$ (**D**'s model of **L**'s model of **D**). Model$_{\mathbf{DLD}}$ consists of a state$_{\mathbf{DLD}}$ and an operator$_{\mathbf{DLD}}$ hierarchy. **D** tracks model$_{\mathbf{DLD}}$ by matching operator$_{\mathbf{DLD}}$ predictions with its own actions. Thus, if **D** were to engage in a *beam*$_{\mathbf{D}}$ after a missile firing, it would be **D**'s recursive tracking of *beam*$_{\mathbf{DLD}}$ which would indicate a missile evasion maneuver to model$_{\mathbf{DL}}$. Further nesting of recursive models could lead to model$_{\mathbf{DLDL}}$, model$_{\mathbf{DLDLD}}$, etc.

Tracking a groups of agents requires the execution of the models of the different agents in the group. Thus, to track a group of opponents in Figure 1, **D** can execute its models of different individual opponents. For instance, suppose a second opponent, **J**, joins **L** in attacking **D** and **E**. **D** can track **J**'s actions just as it tracked **L**'s actions, by executing new models, such as model$_{\mathbf{DJ}}$ and model$_{\mathbf{DJD}}$. In addition, it may also execute others such as model$_{\mathbf{DJL}}$, model$_{\mathbf{DLJ}}$, model$_{\mathbf{DLJL}}$ and so on. These models may be important to track **J**'s interactions with **L**.

Unfortunately, this scheme points to an exponential growth in the number of models that an agent needs to execute. In general, for *N* other agents, and *r* levels of nesting the agent 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, the agent may be unable to execute relevant operators from all its models in real-time, possibly jeopardizing its very survival.

action duration, as well as in the preferences among possible actions can be addressed[28].

Thus, optimizations involving some form of selective tracking appear necessary for real-time execution of these models. Yet, such selectivity should not cause an agent to be completely ignorant of critical information that a model may provide (e.g., a pilot agent should not be ignorant of an opponent's missile firing). One optimization that enables such selective tracking is *model sharing*[25]. The overall motivation is that if there is a model$_{\mathbf{y}}$ that is near-identical to a model$_{\mathbf{x}}$, then model$_{\mathbf{y}}$'s states and operators can be shared with those of model$_{\mathbf{x}}$. Thus, model$_{\mathbf{y}}$ is tracked via the execution of model$_{\mathbf{x}}$, reducing the tracking effort in half. Model$_{\mathbf{y}}$ may be dynamically unshared from 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 model$_{\mathbf{D}}$ and model$_{\mathbf{DLD}}$ (based on the situation in Figure 1). The operator$_{\mathbf{D}}$ hierarchy can be shared with the operator$_{\mathbf{DLD}}$ hierarchy since the two are often identical. Furthermore, information in state$_{\mathbf{D}}$, such as radar input, is shared with state$_{\mathbf{DLD}}$. Thus, **D** essentially executes operators from only one model, instead of two.[4]

There are basically two categories of models that may be shared in this fashion:

1. Models of distinct agents within a group: Agents that are part of a single group may act in a coordinated fashion — executing similar behaviors — providing an opportunity for model sharing. Thus, if a group of agents, say **L** and **J**, together attack **D** in Figure 1, model$_{\mathbf{DL}}$ and model$_{\mathbf{DJ}}$ may possibly be shared. In fact, if models of all of the agents in a group are shared, an agent may execute only one model for the entire group.
2. Recursive models of a single agent at $r \geq 3$: 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. For instance, model$_{\mathbf{DLD}}$ may be shared with model$_{\mathbf{D}}$. If all such models are shared, an agent may need to track no models at $r \geq 3$.

Thus, sharing could provide substantial benefits in tracking. In the best case, an agent may recursively track a group of N agents with just one or two models instead of O(N^r) models. An agent's architecture should thus provide support for model sharing. In addition, the architecture should allow for unsharing of models when they grow dissimilar. For instance, model$_{\mathbf{DL}}$ and model$_{\mathbf{DJ}}$ may be initially shared, but they may need to be unshared should **L** and **J** separate out and attack from two sides.

5 Simultaneous Acting and Tracking

The need for simultaneous acting and tracking implies that an architecture should allow parallel execution of an agent's own model (its behaviors) and models of other agents. If an architecture does not directly support such parallelism, it may simulate that by interleaving the execution of multiple models. One other important implication of this simultaneity is that an architecture should facilitate high-bandwidth communication

[4] If there are some unsharable secret aspects of state$_{\mathbf{D}}$, e.g., if **D**'s missile range is a secret, then it will be maintained on state$_{\mathbf{D}}$, but it will not be shared with state$_{\mathbf{DLD}}$.

among agent models. In particular, the simultaneity is often due to the very close interaction among agents — their behaviors continuously influence each other. Thus, changes in behaviors have to be continuously communicated among models. Indeed, in Figure 1, changes in aircraft maneuvers, and their effect on geometry, have to be continuously communicated among agent models[27]. An agent architecture should facilitate such high bandwidth inter-model communication.

6 Implementation

Previous sections have outlined four key requirements for agent architectures to facilitate real-time, dynamic agent tracking. In particular, an agent architecture should have the following capabilities:

1. Execute multiple agent models (in addition to the agent's own behaviors) — these models may be the agent's models of other agents, or recursive models.
2. Dynamically share and unshare multiple models.
3. Execute multiple models in a (pseudo-)simultaneous fashion.
4. Support high-bandwidth inter-model communication.

We have implemented a variant of the Soar integrated architecture[18] that conforms to these requirements. This variant is actually implemented in terms of Soar rules, so that it forms an interpretive layer on top of Soar. It simulates parallelism in the execution of multiple models (and the agent's own behaviors) via interleaved model execution. We have implemented pilot agents for fighter jets and helicopters (see scenarios in Section 2) based on this architecture.[5] In the following, these agents will be referred as pilot^{tracker} agents. Each pilot^{tracker} agent contains about 1250 rules.

To attempt to evaluate the performance of these agents, we have run the pilots^{tracker} agents in several fighter-jet combat simulation scenarios outlined by our human experts. Figure 4 shows a pilot^{tracker}'s dynamic behavior — the dynamic pattern of time spent in various activities — as it intercepts a single opponent. The X-axis plots time in simulation cycles. Each of these cycle is approximately 66 milliseconds, and the X-axis shows about 3000 cycles. The Y-axis plots the division of architectural activities into cycles for tracking other agent's activities, cycles for the agent's own actions, and wait cycles (in which the pilot^{tracker} waits for the aircraft to complete a turn, or to reach a specified distance from an opponent, without mental activity on its part). The figure shows that much of the initial time is spent waiting. This is because the aircraft are far apart and are flying straight towards each other, with little influence on each other's action. Furthermore, since low-level aircraft control is handled by the simulator, the pilot^{tracker} agent is not mentally occupied. The amount of time spent acting and tracking increases as time passes — basically, the aircraft get closer to each other, and exert an increasing influence on each other's activities.

The key point to note in Figure 4 is that the architecture needs to (and is able to) continuously switch back and forth between acting and tracking activities. Figure 5

[5] These agents are themselves variations of agents, based on Soar, that were developed for this environment[26, 29].

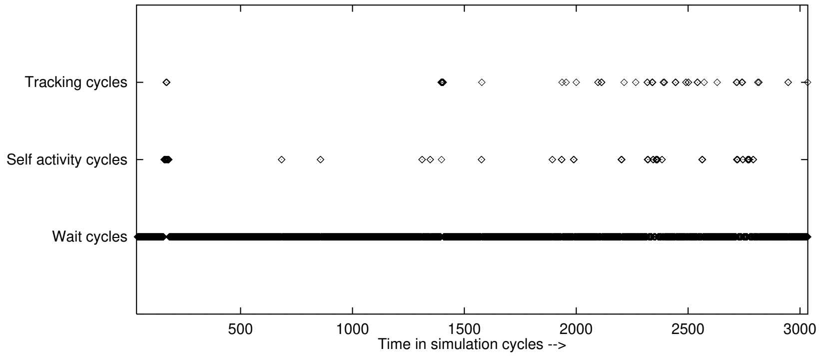

Fig. 4. Dynamic agent behavior in one-vs-one air-combat simulation for a complete engagement. Illustrates simulation cycles devoted by the architecture to tracking, acting, and waiting. Note that one simulation cycle is devoted to one activity.

zooms in on the segment between cycles 2300-2400 from the scenario in Figure 4. It provides a clearer picture of the continuous interleaving between acting and tracking cycles. Further evidence for such interleaving is seen in Figure 6. It shows 100 cycles from another scenario, where a pilot^{tracker} agent is engaged in combat with two opponents, not just one. In this scenario, the architecture devotes even more of its time to tracking. The continuous interleaving in these two scenarios is not a coincidence — each tracking cycle provides an agent information about its opponent, and it quickly reacts given this new information.

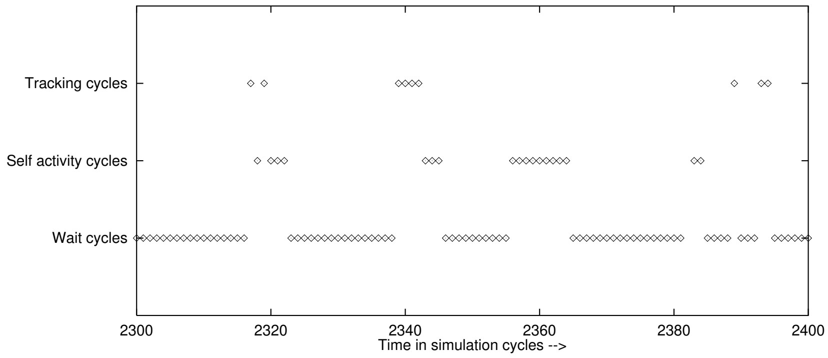

Fig. 5. Dynamic agent behavior in one-vs-one air-combat simulation for 100 cycles.

In both the scenarios above, the pilot^{tracker} agent can accurately track its opponents' actions, in time. Thus, even if the pilot^{tracker} agent makes an initial incorrect inference, it does make a timely correction — to react appropriately.

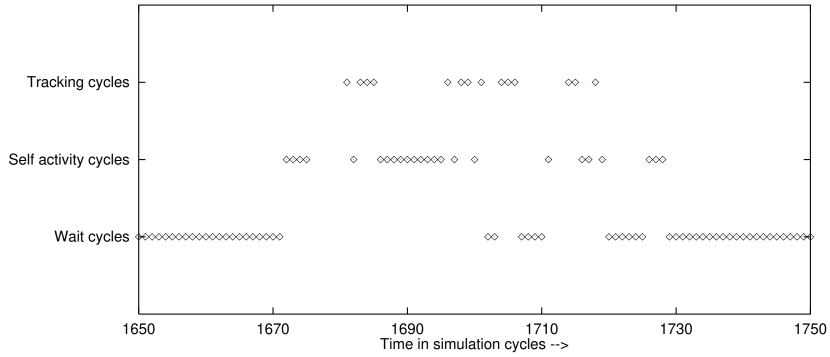

Fig. 6. Dynamic agent behavior in one-vs-two air-combat simulation for 100 cycles.

7 Related Work

There are at least three main areas of related work. The first one is agent architectures. In this area, Rao's work on *reactive plan recognition*[20] based on the PRS/dMARS architecture[10] is closest in spirit to the work reported here. The underlying concept is to extend agent architectures, specifically the PRS architecture, to enable it to execute models of other agents in service of reactive recognition. The resulting architecture can execute an agent's own behavior while simultaneously executing models of other agents in its environment. The similarity goes beyond the proposed architecture — in fact, simulated air-combat is one proposed area of application of this work[21]. While Rao has not focused on recursive agent- and agent-group tracking, it should be possible to extend his work to address those, and apply optimizations such as model sharing discussed in this article. While there are some important differences in the detailed techniques employed — specifically for real-time ambiguity resolution — this basic convergence of architectural extensions is indeed encouraging. In fact, there are at least two other architectures where similar changes have been implemented in service of agent tracking. Hayes-Roth et al. report that their recent work on multi-agent collaborative improvisation has led to similar extensions to their BB1 agent architecture[12]. Specifically, the architectural mechanisms used to plan, control and monitor an agent's own behaviors are reused to monitor, interpret, and predict its partner's behaviors. Ferguson's work on the TouringMachine architecture — focused on agents in dynamic, multi-agent environments — also involves an explicit modeling layer for tracking other agents[9].

A second area of related work is research specifically focused on agent modeling and plan-recognition. Section 1 has discussed some of this work. In addition, some formal approaches for agent modeling, and in particular for recursive agent modeling, are also being investigated[11]. Vidal and Durfee attack the problem of combinatorial explosion in such recursive modeling, and propose a formal approach to tame the combinatorics[31]. Understanding the relationship of these formal approaches to approaches inspired by practical applications, as in the work presented in this article, is a key area for future research.

A third area of related work, also mentioned in Section 1, is research in developmental and cognitive psychology focused on the human ability to ascribe mental states to people: beliefs, desires, and intentions[4]. Baron-Cohen argues that specific neurocognitive mechanisms have evolved to facilitate such mental state ascription, to aid in social understanding, behavioral prediction, social interaction and communication[3]. Selective impairment of these capabilities leads to autism[4]. Such autistic individuals are *mindblind* — their world is devoid of mental things. The implication of this research for AI architectures would appear to be that these architectures should — as is the case with the human cognitive architecture — facilitate agents' ability to reason about the mental states of other agents, or risk agent mindblindess.

8 Conclusion

This article argues that if agents are to successfully inhabit complex, dynamic social worlds, they must obtain architectural support for agent tracking — an important capability required for agent interactions. Key implications of agent tracking for agent architectures include the ability to execute models of other agents, dynamic sharing and unsharing of multiple models, simultaneous (or interleaved) execution of these models, and high bandwidth inter-model communication. We have built an agent architecture, a variant of the Soar integrated architecture, that conforms to these requirements. Agents based on this architecture have been developed for the dynamic, real-time, multi-agent environment of battlefield simulation. While synthetic, this is nonetheless a real-world enviroment, already used in a large-scale operational military exercise[26]. With this variant architecture at its base, agents are capable of successfully tracking others in this challenging environment.

Among issues for future work, we are looking into generalizing the lessons learned here to other real-time comprehension tasks, where the input may not necessarily be other agents' actions. One such task is natural language dialogue, where an agent may need to infer other agent's goals in real-time based on its speech, or language. This will hopefully lead to an improved understanding of the architectural support required for general comprehension capabilities.

9 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).

References

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

2. J. Azarewicz, G. Fala, R. Fink, and C. Heithecker. Plan recognition for airborne tactical decision making. In *Proceedings of the National Conference on Artificial Intelligence*, pages 805–811. Menlo Park, Calif.: AAAI press, 1986.
3. S. Baron-Cohen. *MindBlindness*. MIT Press/AAAI Press, Cambridge, MA, 1995.
4. S. Baron-Cohen, H. Tager-Flusberg, and D. Cohen. *Understanding other minds: perspectives from Autism*. Oxford University Press, Walton Street, Oxford, 1993.
5. J. Bates, A. B. Loyall, and W. S. Reilly. Integrating reactivity, goals and emotions in a broad agent. Technical Report CMU-CS-92-142, School of Computer Science, Carnegie Mellon University, May 1992.
6. M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.
7. R. B. Calder, J. E. Smith, A. J. Courtemanche, J. M. F. Mar, and A. Z. Ceranowicz. Modsaf behavior simulation and control. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*, 1993.
8. J. Cremer, J. Kearney, Y. Papelis, and R. Romano. The software architecture for scenario control in the iowa driving simulator. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*. Orlando, Florida: Institute for Simulation and Training, University of Central Florida, 1994.
9. I. A. Ferguson. *TouringMachines: An architecture for dynamic, rational, mobile agents*. PhD thesis, University of Cambridge, 1992.
10. M. P. Georgeff and A. L. Lansky. Procedural knowledge. *Proceedings of the IEEE special issue on knowledge representation*, 74:1383–1398, 1986.
11. P. Gmytrasiewicz. On reasoning about other agents. In M. Wooldridge, J. Muller, and M. Tambe, editors, *Intelligent Agents, Vol II – Proceedings of the 1995 workshop on Agent theories, Architectures and Languages (ATAL-95)*, Lectures Notes in Articificial Intelligence. Springer-Verlag, Heidelberg, 1996. (In this volume).
12. B. Hayes-Roth, L. Brownston, and Gen R. V. Multiagent collaobration in directed improvisation. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.
13. R. Hill and W. L. Johnson. Situated plan attribution for intelligent tutoring. In *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, Calif.: AAAI press, 1994.
14. M. Huber and E. Durfee. Deciding when to commit to action during observation-based coordination. In *Proceedings of the International Conference on Multi-agent systems (ICMAS)*, 1995.
15. A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the National Conference on Artificial Intelligence*, pages 32–37. Menlo Park, Calif.: AAAI press, 1986.
16. Y. Kuniyoshi, S. Rougeaux, M. Ishii, N. Kita, S. Sakane, and M. Kakikura. Cooperation by observation: the framework and the basic task pattern. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1994.
17. P. Maes, T. Darrell, B. Blumberg, and S. Pentland. Interacting with animated autonomous agents. In J. Bates, editor, *Proceedings of the AAAI Spring Symposium on Believable Agents*, 1994.
18. A. Newell. *Unified Theories of Cognition*. Harvard Univ. Press, Cambridge, Mass., 1990.
19. K. Pimentel and K. Teixeira. *Virtual reality: Through the new looking glass*. Windcrest/McGraw-Hill, 1994.
20. A. S. Rao. Means-end plan recognition: Towards a theory of reactive recognition. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR-94)*.

21. A. S. Rao and G. Murray. Multi-agent mental-state recognition and its application to air-combat modelling. In *Proceedings of the Workshop on Distributed Artificial Intelligence (DAI-94)*. Menlo Park, Calif.: AAAI press, Technical report WS-94-02, 1994.
22. P. S. Rosenbloom, J. E. Laird, A. Newell, , and R. McCarl. A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47(1-3):289–325, 1991.
23. F. Song and R. Cohen. Temporal reasoning during plan recognition. In *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, Calif.: AAAI press, 1991.
24. The DIS steering committee. The dis vision: A map to the future of distributed simulation. Technical report, Institute for simulation and training, University of Central Florida, May 1994.
25. M. Tambe. Recursive agent and agent-group tracking in a real-time dynamic environment. In *Proceedings of the International Conference on Multi-agent systems (ICMAS)*, 1995.
26. M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), Spring 1995.
27. M. Tambe and P. S. Rosenbloom. Event tracking in a dynamic multi-agent environment. *Computational Intelligence*, (To appear), 1995. WWW: http://www.isi.edu/soar/tambe/event.html.
28. M. Tambe and P. S. Rosenbloom. Resc: An approach to agent tracking in a real-time, dynamic environment. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.
29. M. Tambe, K. Schwamb, and P. S. Rosenbloom. Building intelligent pilots for simulated rotary wing aircraft. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 1995.
30. J. A. Thorpe, J. E. Shiflett, G. W. Bloedorn, M. F. Hayes, and D. C. Miller. The simnet network and protocols. Technical Report 7102, BBN systems and technologies corporation, July 1989.
31. J. Vidal and E. Durfee. Recursive agent modeling using limited rationality. In M. Wooldridge, J. Muller, and M. Tambe, editors, *Intelligent Agents, Vol II – Proceedings of the 1995 workshop on Agent theories, Architectures and Languages (ATAL-95)*, Lectures Notes in Articificial Intelligence. Springer-Verlag, Heidelberg, 1996. (In this volume).
32. B. Ward. *ET-Soar: Toward an ITS for Theory-Based Representations*. PhD thesis, School of Computer Science, Carnegie Mellon Univ., 1991.
33. B. Webber and N. Badler. Virtual interactive collaborators for simulation and training. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*. Orlando, Florida: Institute for Simulation and Training, University of Central Florida, May 1993.

This article was processed using the LaTeX macro package with LLNCS style