Agent Architectures for Flexible, Practical Teamwork

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

Abstract

Teamwork in complex, dynamic, multi-agent domains mandates highly flexible coordination and communication. Simply fitting individual agents with precomputed coordination plans will not do, for their inflexibility can cause severe failures in teamwork, and their domain-specificity hinders reusability. Our central hypothesis is that the key to such flexibility and reusability is agent architectures with integrated teamwork capabilities. This fundamental shift in agent architectures is illustrated via an implemented candidate: STEAM. While STEAM is founded on the *joint intentions* theory, practical operationalization has required it to integrate several key novel concepts: (i) *team synchronization* to establish joint intentions; (ii) constructs for monitoring joint intentions and repair; and (iii) decision-theoretic communication selectivity (to pragmatically extend the joint intentions theory). Applications in three different complex domains, with empirical results, are presented.[1]

1 Introduction

...the capabilities needed for collaboration cannot be patched on but must be designed in from the start. —(Grosz 1996)

Teamwork is becoming increasingly critical in many multi-agent environments, such as, virtual training(Tambe *et al.* 1995; Rao *et al.* 1993), interactive entertainment(Hayes-Roth, Brownston, & Gen 1995), internet-based information integration, RoboCup soccer(Kitano *et al.* 1995), and robotic space missions. Teamwork in such complex domains mandates highly flexible coordination and communication to surmount the uncertainties, e.g., dynamic changes in team's goals and team members' unexpected inability to fulfill responsibilities (consider Soccer).

Unfortunately, implemented multi-agent systems often rely on preplanned, domain-specific coordination that fails to provide such flexibility(Jennings 1995). First, it is difficult to anticipate and preplan for all possible coordination failures; particularly in scaling up to complex domains. Second, given domain specificity, reusability suffers — coordination has to be redesigned for each new domain. A fundamental reason for these teamwork limitations is the current agent architectures. Architectures such as Soar(Newell 1990), RAP(Firby 1987), PRS(Rao *et al.* 1993), BB1(Hayes-Roth, Brownston, & Gen 1995), and IRMA(Pollack 1992) facilitate an individual agent's flexible behaviors via mechanisms such as commitments and reactive plans. However, flexible individual behaviors, even if simultaneous and coordinated, do not sum up to teamwork. A common example provided is ordinary traffic, which although simultaneous and coordinated, is not teamwork(Levesque, Cohen, & Nunes 1990). Indeed, theories of collaboration point to fundamentally novel mental constructs as underlying teamwork, such as team goals, mutual beliefs and joint commitments(Grosz 1996; Levesque, Cohen, & Nunes 1990), absent in current agent architectures. Thus, agents cannot explicitly represent and reason about their team goals and plans, or communication/coordination responsibilities in teamwork; instead they rely on the (problematic) preplanned coordination.

Yet, given the ubiquity of teamwork, architectural constructs for flexible, reusable teamwork capabilities are critical. This paper therefore proposes a fundamental shift in agent architectures to integrate teamwork capabilities. It presents one *implemented* candidate: STEAM (a **S**hell for **TEAM**work).[2] Founded on the *joint intentions* theory(Levesque, Cohen, & Nunes 1990), STEAM enables explicit representation of team goals and plans, and team's joint commitments. In practice, to enable multiple team members to maintain a coherent view of their team's goals/plans, STEAM incorporates (i) team synchronization to establish joint intentions; and (ii) monitoring and repair capabilities. Unfortunately, communication in service of coherent teamwork can itself be a significant overhead or risk (in hostile environments). Therefore, STEAM integrates decision theoretic communication selectivity — agents deliberate upon communication necessities vis-a-vis incoherency in teamwork.

2 Illustrative Domains and Motivations

We motivate STEAM by describing key teamwork problems in real-world domains. Two of the domains are based on a real-world battlefield simulator commercially developed for

[2]STEAM code (with documentation/traces) is available at http://www.isi.edu/soar/tambe/steam/steam.html.

military training(Tambe *et al.* 1995). The first domain, Attack (Figure 1), involves pilot agents for a company of (up to eight) attack helicopters. The company typically processes orders and then flies from home-base to a holding point (often in subteams). One or two *scout* helicopters then fly forward and scout the battle position (adjacent to a ridge). Then, other company members fly forward to the battle position. Here, individual pilots repeatedly mask(hide) their helicopters and unmask to shoot missiles at enemy targets. In the second domain, Transport (Figure 2), transports protected by escort helicopters fly troops to land. Our third domain is RoboCup synthetic soccer(Kitano *et al.* 1995).

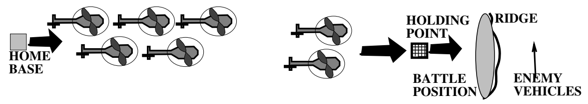


Figure 1: Attack domain: company flying in subteams

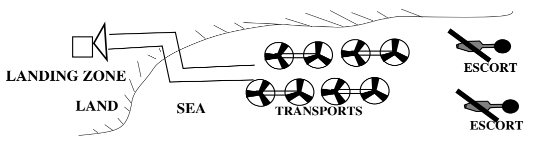


Figure 2: Transport domain.

The Attack domain is illustrative of the teamwork challenges. In our initial implementation, based on Soar, a standard operator hierarchy was defined for each individual pilot. Figure 3 shows a portion of this hierarchy (for now, ignore the brackets [] shown around some operators). Operators are very similar to reactive plans commonly used in other agent architectures. Each operator consists of (i) precondition rules for selection; (ii) rules for application (a complex operator subgoals); and (iii) rules for termination. For teamwork among individuals, domain-specific coordination plans were added (as commonly done in other such efforts). For instance, after scouting the battle position, the scout executes a plan to inform those waiting behind that the battle position is scouted (not shown in Figure 3).

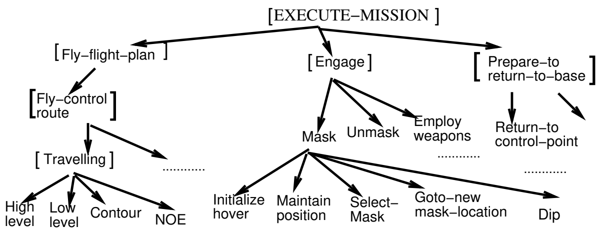


Figure 3: Attack domain: Portion of operator hierarchy.

Unfortunately, despite carefully preplanned coordination, a variety of teamwork failures were encountered. First, pilot agents violated important synchronization points. For instance, once one pilot unexpectedly processed its initial orders before others. It then flew towards the holding point, while its teammates were left behind at the home base. Second, agents sometimes failed to communicate important information to others. For instance, upon abnormally terminating the mission, the commander returned to home base alone without informing others and thus abandoning them at the battle position. Third, agents failed to correctly monitor team performance and "hung" indefinitely. For instance, once only a scout made it to the holding area (all others crashed or got shot down); but the scout scouted the battle position anyhow, and waited indefinitely for its non-existant company to move forward. Conversely, when a lone scout was destroyed, the rest of the company waited indefinitely for its scouting message!

One approach to address these failures is a further addition of domain-specific coordination plans. However, there are several difficulties. First, there is no overarching framework that would enable anticipation of teamwork failures. Instead, coordination plans have to be added on a case-by-case basis — a difficult process, since failures have to be first encountered in actual runs. Furthermore, as the system continues to scale up to increasingly complex teamwork scenarios, such failures continually recur. Thus, a large number of special case coordination plans are potentially necessary. Finally, it is difficult to reuse such plans in other domains.

Given these difficulties, we propose an alternative approach — provide agents with a general model of teamwork. The agents can then themselves reason about their coordination/communication responsibilities and avoid teamwork failures. Such an approach requires that the underlying architecture facilitate agents' explicit representation and reasoning with team goals/plans, which it at present does not. In particular, in the current architecture (Soar), an agent's operator hierarchy represents only its current activities, and the agent is often ignorant as to which operators involve teamwork (and the teammates involved in them). For instance, *execute-mission* and *engage* are in reality team activities involving the entire company; while *mask* and *unmask* involve no teamwork. Indeed, some team tasks — e.g., scouts scouting the battle position while the non-scouts wait — can not be represented in the operator hierarchy. Furthermore, the architecture does not incorporate a model of the commitments or coordination responsibilities involved in team activities. Thus, agents are forced to rely on the problematic domain-specific coordination plans. This issue is not specific to the Soar architecture, but the entire family of architectures mentioned in Section 1.

3 STEAM Foundations: Joint Intentions

STEAM is founded on the *joint intentions* theory(Levesque, Cohen, & Nunes 1990). A joint intention of a team Θ is based on its joint commitment, which is defined as a joint persistent goal (JPG). A JPG to achieve a team action **p**, denoted JPG(Θ, **p**) requires all teammembers to mutually believe that **p** is currently false and want **p** to be eventually true.

JPG provides a basic change in plan expressiveness, since it focuses on a team task. Furthermore, a JPG guarantees that team members cannot decommit until **p** is mutually known to be achieved, unachievable or irrelevant. Basically, JPG(Θ, **p**) requires team members to each hold **p** as a weak achievement goal (WAG).[3] WAG(μ, **p**, Θ), where μ is a team member in Θ, requires μ to achieve **p** if it is false. However, if μ privately believes that **p** is either achieved, unachievable or irrelevant, JPG(Θ,**p**) is dissolved, but μ is left with a commitment to have this belief become Θ's mutual belief. Such a commitment avoids key communication failures presented in Section 2 — to establish mutual belief, an agent must typically communicate with its teammates.

Members of Θ must synchronize to establish JPG(Θ,**p**). To achieve such *team synchronization* we adapt the *request-confirm* protocol(Smith & Cohen 1996), described below. The key here is a persistent weak achievement goal (PWAG(νi, **p**, Θ)), which commits a team member νi to its team task **p** prior to a JPG. μ initiates the protocol while its teammates in Θ, ν1,.,νi..νn, respond:

1. μ executes a **Request**(μ, Θ, p), cast as an **Attempt**(μ, ϕ, ψ). That is, μ's ultimate goal ϕ is to both achieve **p**, and have all νi adopt PWAG(νi, **p**, Θ). However, μ is minimally committed to ψ, i.e., just to achieve mutual belief in Θ that μ has the PWAG to achieve ϕ. With this **Request**, μ adopts the PWAG.
2. Each νi responds via **confirm** or **refuse**. **Confirm**, also an **Attempt**, informs others that νi has the PWAG to achieve **p**.
3. If $\forall$ i, νi confirm, JPG(Θ, **p**) is formed.

Besides synchronization, this protocol enforces important behavioral constraints. In step 1, the adoption of a PWAG implies that if after requesting, μ privately believes that **p** is achieved, unachievable or irrelevant, it must inform its teammates. Furthermore, if μ believes that the minimal commitment ψ is not achieved (e.g., the message did not get through) it must retransmit the message. Step 2 similarly constrains team members νi to inform others about **p**, and to rebroadcast. If everyone confirms, a JPG is established.

4 The STEAM Architecture: Basics

STEAM's basis is in executing hierarchical reactive plans, in common with architectures mentioned in Section 1. One key architectural novelty in STEAM is *team operators* (reactive team plans), which instantiate a team's joint intentions, and thus facilitate reasoning about teamwork. Team operators explicitly express a team's joint activities, unlike the regular "individual operators" which express an agent's own activities. In the hierarchy in Figure 3, operators shown in [] such as [Engage] are team operators (others are individual operators). As with individual operators, team operators also consists of: (i) precondition rules; (ii) application rules; and (iii) termination rules. STEAM maintains a private state for an agent to apply its individual operators; and a "team state" to apply team operators. A team state is an agent's (abstract) model of the team's mutual beliefs about the world, e.g., in the Transport domain, the team state includes the coordinates of the landing zone. STEAM can also maintain subteam states for subteam participation.

Given an arbitrary team operator OP, all team members must simultaneously select OP to establish a joint intention (joint intention for OP will be denoted as $[\mathrm{OP}]_\Theta$). To this end, agents execute the following team synchronization procedure based on the request-confirm protocol:

1. Team leader broadcasts a message to the team Θ to establish PWAG to operator OP. Leader now establishes PWAG. If $[\mathrm{OP}]_\Theta$ not established within time limit, repeat broadcast.
2. Subordinates νi in the team wait until they receive leader's message. Then, turn by turn, broadcast to Θ establishment of PWAG for OP; and establish PWAG.
3. Wait until $\forall$ νi, νi establish PWAG for OP; establish $[\mathrm{OP}]_\Theta$.

With this protocol, agents avoid synchronization problems of the type where just one member flies off to the holding area. During execution, PWAGs address several contingencies — if OP is believed achieved, unachievable or irrelevant prior to $[\mathrm{OP}]_\Theta$, agents inform teammates. Other contingencies are also addressed, e.g., even if a subordinate initially disagrees with the leader, it will conform to the leader's broadcasted choice of operator in synchronization.

After team synchronization, a team operator can only be terminated by updating the team state (mutual beliefs). This restriction on team operator termination avoids critical communication failures of the type where the commander returned to home-base alone — instead, agents must now inform teammates when terminating team operators. That is, if an agent's private state contains a belief that terminates a team operator — makes it achieved, unachievable, or irrelevant — and such a belief is absent in its team state, then it creates a communicative goal, i.e., a communication operator. This operator broadcasts the belief to the team, updating the team state, and then terminating the team operator. This broadcast also communicates the cause of termination so team members can decide their next action. For instance, if company member ν4 sees some tanks on the route, it recognizes that this fact causes the team's current joint intention $[\text{fly-flight-plan}]_\Theta$ to be unachievable. If this fact is absent in the team state, then a communication operator is executed, resulting in a message broadcast, e.g., *ν4 terminate-JPG fly-flight-plan evade tank 61000 41000*. This informs team members to terminate $[\text{fly-flight-plan}]_\Theta$ and provides a reason (evade tank at x,y location). Thus, ν4 informs others; it does not evade tanks on its own.

5 Monitoring and Repair

One major source of teamwork failures, as outlined in Section 2, is agents' inability to monitor team performance. STEAM facilitates such monitoring by exploiting its explicit representation of team operators. In particular, STEAM allows an explicit specification of monitoring conditions to determine achievement, unachievability or irrelevancy of team operators. Furthermore, STEAM facilitates explicit specification of the relationship between a team operator

[3] WAG was originally called WG in (Levesque, Cohen, & Nunes 1990), but later termed WAG in (Smith & Cohen 1996).

and individuals' *roles* in it. A *role* constrains a team member νi to some suboperator op$_{\nu i}$ of the team operator [OP]$_\Theta$. Currently the following key relationships — called *role-monitoring constraints* — can be specified in STEAM:

1. *AND-combination*: [OP]$_\Theta \Longleftrightarrow \bigwedge_{i=1}^{n}$ op$_{\nu i}$
2. *OR-combination*: [OP]$_\Theta \Longleftrightarrow \bigvee_{i=1}^{n}$ op$_{\nu i}$
3. *Role dependency*: op$_{\nu i} \Longrightarrow$ op$_{\nu j}$ (op$_{\nu i}$ dependent on op$_{\nu j}$)

STEAM uses these specifications to infer the achievement or unachievability of a team operator based on individual's role performance. Combinations of the above relationships can also be specified. For example, for two agents νi and νj, with roles op$_{\nu i}$ and op$_{\nu j}$, both an OR-combination plus role-dependency may be specified as ((op$_{\nu i} \bigvee$ op$_{\nu j}$) $\bigwedge$ (op$_{\nu i} \Longrightarrow$ op$_{\nu j}$)). STEAM can now infer that the role non-performance of νj ($\neg$op$_{\nu j}$) makes [OP]$_\Theta$ unachievable; but role non-performance of νi is not critical to [OP]$_\Theta$.

When a team operator [OP]$_\Theta$ becomes unachievable, STEAM invokes [repair]$_\Theta$ for repair. By casting repair as a team operator, agents automatically synchronize the execution of [repair]$_\Theta$ and inform teammates about repair results. The actions taken in service of [repair]$_\Theta$ depend on the context. If [repair]$_\Theta$ was invoked due to [OP]$_\Theta$'s unachievability conditions, domain-specific repair is triggered. In contrast, role-monitoring constraint failure causes STEAM to first analyze the failure. The analysis may reveal a *critical role failure* — a single role failure causing the unachievability of [OP]$_\Theta$ — which may occur in an AND-combination if any agent fails in its role; or an OR-combination when all team members are role-dependent on a single individual. For instance, when agents are flying in formation via [travelling]$_\Theta$ (OR-combination), everyone is role-dependent on the lead helicopter. Thus, should the lead crash, a critical role failure occurs. The action taken in such cases is team reconfiguration, where an agent capable of performing the critical role takes over that role. Since [repair]$_\Theta$ is a team operator, this repair is announced to Θ. In contrast, a pure *role dependency failure* disables only a single dependent agent νi from role-performance (because op$_{\nu i} \Longrightarrow$ op$_{\nu j}$). Here, νi must locate another agent νk such that op$_{\nu i} \Longrightarrow$ op$_{\nu k}$. If failure type is *all roles failure*, no agent performs its role; which is irreparable. ([repair]$_\Omega$ may be invoked for a subteam Ω of Θ — repair communication is then automatically restricted within Ω).

If [repair]$_\Theta$ is itself unachievable — no repairs achieved within time limit — [complete-failure]$_\Theta$ is invoked, to act given complete failure of [OP]$_\Theta$. For instance, in the Attack domain, complete failure implies returning to home base.

6 Selective Communication

STEAM agents communicate for team operator synchronization and termination. Given the large number of team operators in a dynamic environment, this communication is a very significant overhead (as Section 7 shows empirically), or risk (e.g., in hostile environments).

Therefore, STEAM integrates decision theoretic communication selectivity, pragmatically extending the joint intentions framework — agents may not *guarantee* mutual beliefs when terminating joint intentions. The key is not only communication costs and benefits, but the *likelihood that relevant information is already common knowledge* and hence unnecessary to communicate. Figure 4 presents the decision tree for the decision to communicate a fact F, indicating the termination of [OP]$_\Theta$. Rewards and costs are measured to the team, not an individual. The two possible actions are **NC** (not communicate, cost 0) or **C** (communicate, cost Cc). If the action is **NC**, two outcomes are possible. With probability (1-τ), F was commonly known anyway, and the team is rewarded *B* for terminating [OP]$_\Theta$. With probability τ, however, F was not known, and thus there is miscoordination in terminating [OP]$_\Theta$ (e.g., some agents come to know of F only later). Given a penalty C$_{mt}$ for miscoordination, the reward reduces to *B*-*C*$_{mt}$. If the action is **C**, assuming reliable communication, F is known.

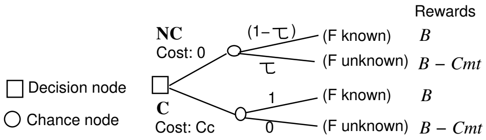


Figure 4: Decision tree for communication.

EU(**C**), the expected utility of option **C**, is *B*-*Cc*. EU(**NC**) of option **NC** is *B*-τ*C_{mt}. To maximize expected utility, an agent communicates iff EU(**C**) $>$ EU(**NC**), i.e., iff τ*C$_{mt}$ $>$ Cc. Thus, for instance, in the Attack domain, when flying with high visibility, pilot agents do not inform others of achievement of waypoints on their route, since τ is low (high likelihood of common knowledge), and C$_{mt}$ is low (low penalty). However, they inform others about enemy tanks on the route, since although τ is low, C$_{mt}$ is high.

Expected utility maximization is also used for selectivity in team synchronization messages. If γ is the probability of lack of team synchronization and C$_{me}$ the penalty for executing [OP]$_\Theta$ without synchronization, then an agent communicates iff EU(**C**) $>$ EU(**NC**), i.e., γ* C$_{me}$ $>$ Cc.

6.1 Further Communication Generalization

Further generalization in communication is required to handle uncertainty in the termination criteria for joint intentions. For instance, a team member ν4 may be uncertain that an enemy tank seen enroute causes [fly-flight-plan]$_\Theta$ to be unachievable — the tank's threat may not be clearcut. Yet not communicating could be highly risky. The decision tree for communication is therefore extended to include δ, the uncertainty of an event's threat to the joint intention (Figure 5). Since agents may now erroneously inform teammates to terminate team operators, a nuisance cost -*Cn* is introduced.

Again, an agent communicates iff EU(**C**) $>$ EU(**NC**), i.e., iff δ*τ*C$_{mt}$ $>$ (Cc + (1-δ)Cn). If δ is 1, i.e., a team operator has terminated, this equation reduces to — τ*C$_{mt}$ $>$ Cc — seen previously. If δ $<<$ 1, i.e., high uncertainty about termination, no communication results if Cn is high. Therefore, the decision tree is further extended to include a new message type — threat to joint intention — where Cn

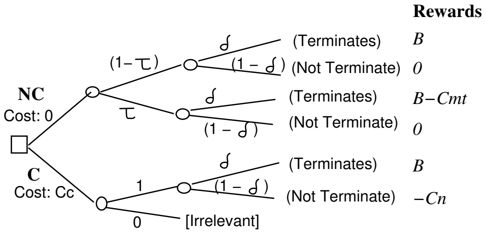


Figure 5: Extended decision tree with δ.

is zero, but benefits accrued are lower (B - C_ϵ). This threat message maximizes expected utility when $\delta << 1$, i.e., if Cn is high for communicating termination, a team member communicates a threat. For instance, a threat message is used if an agent fails in its own role, which is a threat to the joint intention. However, as before, termination messages are used when δ = 1, where they maximize expected utility.

6.2 Estimating Parameters (γ, τ, δ)

As a first step, STEAM only uses qualitative (low, high, medium) parameter values. STEAM estimates likelihood of lack of team synchronization, γ, via team tracking(Tambe 1996b) — dynamically inferring a team's mental state from observations of team members' actions. Fortunately, rather than tracking each teammate separately, an agent νi can rely on its own team operator execution for team tracking. In particular, suppose νi has selected a team operator OP for execution, and it needs to estimate γ for synchronization with team Θ. Now, if νi selected OP at random from a choice of equally preferable candidates, then its teammates may differ in this selection. Thus, there is clearly a low likelihood of team synchronization — νi estimates γ to be high. However, if OP is the only choice available, then γ depends on the preceding $[OP2]_\Omega$ that νi executed with the team Ω. There are three cases to consider. First, if $\Theta \subset \Omega$ (Θ is subteam of Ω) or $\Theta = \Omega$, all members of Θ were synchronously executing $[OP2]_\Omega$. Thus, Θ is likely synchronized in executing the only next choice OP — γ is estimated low. Second, if $\Omega \subset \Theta$, some members in Θ were not synchronized earlier; hence γ is estimated high. Third, if no operator precedes OP (e.g., OP is first in a subgoal) then γ is estimated low.

While agents usually infer matching estimates of γ, sometimes, estimates do mismatch. Therefore, STEAM integrates some error recovery routines. For instance, if an agent νi estimates γ to be low, when others estimate it high, νi starts executing the team operator, and only later receives a message for team synchronization. νi recovers by stopping current activities and synchronizing. In contrast, if νi mis-estimates γ to be high, it unnecessarily waits for synchronization. STEAM infers such a mis-estimation via reception of unexpected messages; it then conducts a lookahed search to catch up with teammates.

To estimate τ, STEAM assumes identical sensory capabilities for team members, e.g., if some fact is visible to an agent, then it is also visible to all colocated teammates. δ is estimated 1 if a fact matches specified termination conditions. Otherwise role monitoring constraints are used, e.g., in an OR-combination, δ is inversely proportional to the number of team members. The cost parameters, C_{mt}, C_{me}, and Cc are assumed to be domain knowledge.

7 Evaluation

STEAM is currently implemented within Soar via conventions for encoding operators and states, plus a set of 251 rules. STEAM-based pilots in the Attack and Transport domains have participated in several large-scale synthetic military exercises with hundreds of agents. Here, experts (human pilots) have set the tests for these pilot teams and issued favorable performance evaluations.

STEAM's flexibility and reusability is approximately measured in Table 1. Column 1 lists three different domains of STEAM's application — Attack, Transport, RoboCup — thus providing some evidence of STEAM's generality. Column 2 lists the total number of rules per domain, illustrating domain complexity. Column 3 lists the number of STEAM rules used in these domains; while column 4 measures percent reuse of STEAM rules across domains. (No reuse is shown in STEAM's first domain, Attack). There is 100% reuse in Transport, i.e., no new coordination/communication rules were written — a major saving in encoding this domain. RoboCup, in its initial stages, has lower reuse. Here, due to weakness in spatial reasoning and tracking, agents fail to recognize other team's play, or even own teammates' failures (e.g., in executing a pass), hampering the reuse of rules for role-monitoring constraints, repair and threat detection. With improved spatial reasoning and tracking, reuse may improve in the future.

Domain	Total rules	STEAM rules	STEAM reuse	Team opers
Attack	1466	251	first-use	17
Transport	1303	251	100%	14
RoboCup	202	100	40%	5

Table 1: Reusability and flexibility data.

Column 5 lists the total number of team operators specified per domain. For each team operator STEAM's entire teamwork capabilities are brought to bear. As a result, in benchmark runs of Attack, almost all of the teamwork failures from our earlier implementation are avoided. Thus, considerable flexibility is achieved without the cost of adding many special case coordination plans, e.g., hundreds special case operators would be needed in our initial implementation just to reproduce STEAM's decision-theoretic communication selectivity. Additionally, STEAM appears to more easily accommodate modifications suggested by domain experts; at least, no new coordination plans are required. For instance, in the Attack domain, domain experts earlier suggested a modification, that the helicopter company should evade enemy vehicles seen enroute, rather than flying over. Here, adding a new unachievability condition for the team operator $[\text{fly-flight-plan}]_\Theta$ was sufficient; STEAM then ensured that the pilot agents coordinated the termination of [fly-flight-plan]$_{\Theta}$, even if just one arbitrary team member detected the enemy vehicles. (Of course, the evasion maneuvers, being domain-specific, had to be added.)

Figure 6 illustrates the usefulness of STEAM's decision-theoretic communication selectivity. It compares the total number of messages in three teams — balanced, cautious and reckless — with increasing numbers of agents per teams. Balanced agents fully exploit the decision theory framework. Cautious agents always communicate, ignoring the decision theory framework. Reckless agents communicate very little (only if high C_{mt}, C_{me}). All three teams work with identical cost models, Cc, C_{mt}, and C_{me}. Decision-theoretic selectivity enables the balanced team to perform well with few messages — this team is regularly fielded in synthetic exercises. The cautious team exchanges 10 to 20-fold or more messages than the balanced team — a substantial communication overhead. Indeed, beyond six agents, the simulation with cautious team could not be run in real time. Reckless agents do exchange fewer messages, but they miscoordinate, e.g., team members are stranded throughout the battlefield.

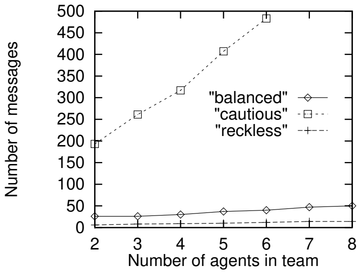
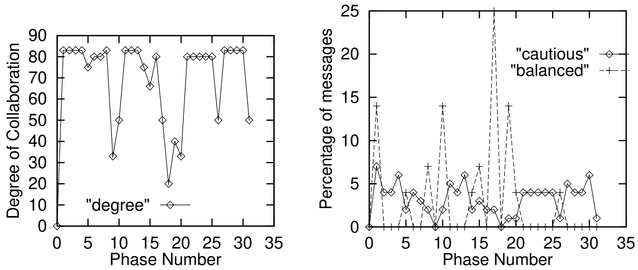


Figure 6: Attack domain: selective communication.



(a) Degree of collaboration (b) Percentage communication

Figure 7: Attack domain: pattern of communication

Figures 7 illustrate the differing communication patterns in the cautious and balanced teams. Figure 7-a plots the varying degree of collaboration (y-axis) during different phases (x-axis) in the Attack domain. Degree of collaboration is measured as the percentage of team operators in a pilot's operator hierarchy (which consists of team and individual operators). A low percentage implies low degree of collaboration and vice versa. The degree of collaboration is lowest in phases 18-20 (20-40%), where agents engage the enemy. Figure 7-b plots the *percentage* of total communication per phase, for cautious and balanced teams. Communication percentage is correlated to the degree of collaboration per phase for the cautious team (coefficient 0.80), but not for the balanced team (coefficient -0.34).

8 Summary and Related Work

Motivated by the critical need for teamwork flexibility and reusability, this paper presents a novel architecture with integrated teamwork capabilities. Key contribution include: (i) an architecture, STEAM, founded on joint intentions; (ii) applying decision theory for communication selectivity and enhancements within the joint intentions framework; (iii) monitoring and repair method based on explicit team operators; (iv) integration of team synchronization; (v) STEAM's implementation and application in three complex domains; indeed, pilots for Attack and Transport have participated in real-world synthetic exercises with hundreds of agents.

Several major issues remain open for future work. One key issue is investigating STEAM's interactions with learning. Initial experiments with chunking (a form of explanation-based learning) in STEAM reveal that agents could automatize routine teamwork activities, rather than always reasoning about them. Specifically, from STEAM's domain-independent reasoning about teamwork, agents learn situation-specific coordination rules. For instance, when the formation leader crashes, another agent learns situation-specific rules to take over as formation lead and communicate. Additionally, STEAM's knowledge-intensive approach could complement current inductive learning approaches for multi-agent coordination(Sen 1996).

Failure detection and recovery is also a key topic for future work, particularly in environments with unreliable communication. One novel approach exploits agent tracking(Tambe & Rosenbloom 1995; Tambe 1996b) to infer teammates' high-level goals and intentions for comparison with own goals and intentions. Differences in goals and intentions may indicate coordination failures, since teammates often carry out identical or related tasks. See (Kaminka & Tambe 1997) for more details.

Among related work, STEAM is closely related to the recent model-based collaborative systems(Jennings 1995; Tambe 1996a). These systems provide agents with a model of teamwork based on joint intentions. However, such models require the architectural moorings of explicit team goals, team states, etc. This fact, plus the ubiquity of teamwork, has led to STEAM's architectural approach. Also novel in STEAM is the illustrated reuse across domains. Furthermore, STEAM handles more complex team organizations, including subteams. Finally, STEAM substantially extends the teamwork models. For instance, while (Jennings 1995) supports team synchronization (although without the benefit of PWAGs) it does not address communication selectivity, or role-monitoring constraints and integrated role repair. (Tambe 1996a) does not consider team synchronization; and furthermore, while raising the issue of communication cost, suggests only a heuristic evaluation of communication costs and benefits. In contrast, STEAM considers a variety of uncertainties within the decision theory framework. Similarly (Tambe 1996a) can only monitor the loss of one "specialist" in a team, and repair that specific failure. In contrast, STEAM can monitor a greater variety of failures, and it integrates repair within the teamwork framework.

While STEAM is also related to COLLAGEN(Rich & Sidner 1997) — another recent system based on model-based collaboration — the two systems emphasize complementary capabilities. In particular, COLLAGEN focuses on discourse and interactions between a human user and an intelligent agent, rather than multi-agent interaction. Thus, discourse generation and interpretation is important in COLLAGEN, but capabilities critical in STEAM such as teamwork monitoring and repair, or decision-theoretic communication selectivity are at present excluded. An additional interesting contrast is that while COLLAGEN is based on the SharedPlan theory of collaboration(Grosz 1996), STEAM is based on the joint intentions theory. A comparative analysis of the two systems for possible cross-fertilization of ideas/techniques is an issue for future work.

In team tracking(Tambe 1996b), i.e., inferring team's joint intentions, the expressiveness of team operators has been exploited. However, issues of synchronization, communication, monitoring and repair are not addressed. The formal approach to teamwork in (Sonenberg *et al.* 1994) transforms team plans into separate role-plans for execution by individuals, with rigidly embedded communications. STEAM purposely avoids such transformations, so agents can flexibly reason with (i) explicit team goals/plans; and (ii) selective communication (very important in practice). In (Gmytrasiewicz, Durfee, & Wehe 1991), decision theory is applied for message prioritization in coordination (rather than teamwork). STEAM applies decision theory in a very different context — joint intentions — for communication selectivity and enhancements. Also, STEAM integrates other key facilities for teamwork, and implements this theory in real-world domains.

Acknowledgements

Comments from Paul Rosenbloom, Ramesh Patil, John Laird, Yolanda Gil, Kevin Knight and Jeff Rickel helped in significantly improving the quality of this paper. Teamwork with Jon Gratch, Randy Hill and Johnny Chen was key in development of pilot agent teams. This research was supported as part of contract N66001-95-C-6013 from ARPA/ISO.

References

Firby, J. 1987. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

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*.

Grosz, B. 1996. Collaborating systems. *AI magazine* 17(2).

Hayes-Roth, B.; Brownston, L.; and Gen, R. V. 1995. Multiagent collaobration in directed improvisation. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*.

Jennings, N. 1995. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence* 75.

Kaminka, G., and Tambe, M. 1997. Social comparison for failure monitoring and recovery in multi-agent settings. In *Proceedings of the National Conference on Artificial Intelligence*, (Student abstract).

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; and Osawa, E. 1995. Robocup: The robot world cup initiative. In *Proceedings of IJCAI-95 Workshop on Entertainment and AI/Alife*.

Levesque, H. J.; Cohen, P. R.; and Nunes, J. 1990. On acting together. In *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, Calif.: AAAI press.

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

Pollack, M. 1992. The uses of plans. *Artificial Intelligence* 57:43–68.

Rao, A. S.; Lucas, A.; Morley, D.; Selvestrel, M.; and Murray, G. 1993. Agent-oriented architecture for air-combat simulation. Technical Report Technical Note 42, The Australian Artificial Intelligence Institute.

Rich, C., and Sidner, C. 1997. COLLAGEN: When agents collaborate with people. In *Proceedings of the International Conference on Autonomous Agents (Agents'97)*.

Sen, S. 1996. *Proceedings of the Spring Symposium on Adaptation, Coevolution and Learning*. Menlo Park, CA: American Association for Artificial Intelligence.

Smith, I., and Cohen, P. 1996. Towards semantics for an agent communication language based on speech acts. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Sonenberg, E.; Tidhard, G.; Werner, E.; Kinny, D.; Ljungberg, M.; and Rao, A. 1994. Planned team activity. Technical Report 26, Australian AI Institute.

Tambe, M., and Rosenbloom, P. S. 1995. RESC: An approach for real-time, dynamic agent tracking. 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).

Tambe, M. 1996a. Teamwork in real-world, dynamic environments. In *Proceedings of the International Conference on Multi-agent Systems (ICMAS)*.

Tambe, M. 1996b. Tracking dynamic team activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.