

TEAMBOTICA: a robotic framework for integrated teaming, tasking, networking, and control

Regis Vincent, Pauline Berry, Andrew Agno, Charlie Ortiz, David Wilkins
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025

{vincent, berry, agno, ortiz, wilkins}@ai.sri.com

ABSTRACT

TEAMBOTICA is a research environment for the exploration of theories, designs and implementations of team-based robotics. In developing TEAMBOTICA, we found that many of the simplifying assumptions that are often taken in both multiagent systems and behavior-based robotics had to be discarded. Central to our approach is a multilevel agent architecture which is adaptive along a number of dimensions and which is based on a vertically integrated design that spans a wide range of operations, from team-level reasoning to low-level control. The design addresses a number of pertinent issues: the proper mix of deliberation and action, flexible networking support including planning for communications, adaptive task level control, team-based monitoring, and an open systems modularity that takes form-factor considerations seriously. We also describe simulation tools for development and discuss several robotic teams that we have demonstrated.

Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous;
D.2.8 [Software Engineering]: Metrics—*complexity measures, performance measures*

Keywords

Team robotics, hybrid architectures, collaboration.

1. INTRODUCTION

We describe an ongoing research program to develop an environment, which we call TEAMBOTICA, for the exploration of theories, designs and implementations of team-based robotics. One of the lessons that has emerged from our work is the realization that many of the simplifying assumptions that are often taken in both multiagent systems and behavior-based robotics must be discarded. While these

assumptions have allowed progress in individual areas, when taken as a whole, they produce impediments to the realization of flexible, physically embodied robotic teams.

For example, team-based robotic systems have tended to focus on only a subset of key functionalities at the expense of a vertically-integrated design. One example is that of behavior-based robotic designs which typically neglect deliberation in favor of rapid reaction. The result is an inability to consider the long term consequences of actions. Hybrid architectures address this shortcoming but have been restricted to designs for individual robots. Systems that are grounded in rich theories of collaboration have often been tested only in simulation, thereby neglecting issues in low-level control, communication and adaptivity to task failure. On the other hand, team-level robotic systems, such as Robocup, focus on reaction—primarily as a consequence of the reactive requirements of the chosen problem domain—and do not incorporate rich notions of collaboration. Research in communication for multiagent systems has focussed on the semantic content of messages and not on a flexible underlying network that could support communication: this is important for robotic teams since individual elements of the team might enter areas in which communications are blocked by some physical structure (although there are systems that factor in the cost of communication, adaptivity at the network level is never addressed). Discarding any of these assumptions can introduce further challenges if one is also faced with form factor and design modularity concerns.

In this paper, we describe progress toward developing a vertically-integrated framework which is adaptive at the team, task, network, and control levels. In our work, we have not pursued emergent behavior approaches. Although we find such work enormously interesting, we favor instead approaches that allow more control over the development and debugging process.

The remainder of this paper is structured in the following way. We begin by a discussion of a multilevel agent architecture which forms the basis for TEAMBOTICA. We then go on to describe our contributions in key supporting technologies: (1) execution task control, (2) negotiation and collaboration, (3) monitoring, (4) ad-hoc mobile networking; (7) open systems experimental framework; and (8) team-based robotic systems that we have demonstrated.

2. PROBLEM

The assumptions we found we had to address follow:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

- *Communication is instantaneous and assured.* This assumption is that robots can exchange all the information they want, when they want. This is not the case in real life: wireless systems are very common now but they rely on either a base station to forward the message between the robots, or ad-hoc, single hop, line-of-sight communications. Neither of these is particularly robust when robots are navigating around obstacles that may block radio waves. In our case, there is no base station, and no single hop assumption. A robot relies on teammates to relay messages. This avoids single points of failure and avoids having to install access points around the field of operations.
- *Actions always perform as predicted.* Here, we have the assumption that a robot's actions are successful. Oftentimes, with physical robots, actions are not successful: they are not performed as we expect, or are not performed at all. almost all the robotic actions were diverging significantly from our predictions. While this may indicate a poor prediction model, real robots present the problem that it is hard to determine the variables for a model. Even if one could identify all the variables, not all variables are easily quantified. We had to deal with the fact that actions are unpredictable and yet we had to make some estimates. In the worst and most common case, actions simply fail. Monitoring actions was the key to solving this problem.
- *Goals are not modified during deliberation or execution.* It is much easier for a robot to consider all its goals before taking any action, and then to execute the planned actions. In a real environment, however, goals vary and may need to be modified over time.

3. MOTIVATING SCENARIO

The current scenario we have implemented and tested using physical robots is as follows. A team of two robots and one unmanned aerial vehicle (UAV) are tasked to patrol and track any intruders in an area. The robots negotiate to split the area into 3 zones. Each evaluates where they would be optimally useful based on its capabilities. Each then proposes a solution to others and resolves conflicts (see section 5.3 for details).

Once the team members have adopted individual intentions to patrol specific areas, they deploy and start patrolling using their own algorithms. When a robot detects an intruder, it activates the tracking task. Now this robot tries to detect any other intruders while tracking. If the intruder exits its patrolling area, the robot pauses its patrolling task to concentrate on the tracking. Since it is outside of its *assigned* area, it will ask for a handoff. The computation of the handoff is based on a weighted function between the robot's intercepting time, current tasks, and assigned area. If another robot is more appropriate to track the evader, there will be an handoff where the second robot will intercept the evader and release the first robot of its duty. The first robot will then resume its patrolling task and return to its assigned area.

4. MULTILEVEL ARCHITECTURE

Teams are comprised of autonomous vehicles (AVs). Each AV is architected to reflect two dimensions of organization: a functional dimension and a software dimension. The former segments robot functionality into what we refer to as team, strategy, tactical, and control levels. Figure 1 illustrates the architecture. Besides associating each level with a particular functionality, the complexity of each computation that takes place at any particular level is conceptually bounded, reflecting the expected time available for decisions made at that level. The lowest levels are responsible for purely reactive robot behavior, while more deliberative and goal-directed behavior takes place at the higher levels, generally over a longer period of time. At the control level, response is immediate; the robot has a 10ms cycle time for responses. At the task level, responses take place within $\frac{1}{2}$ to 1s. Computations taking place at the strategy level generally take 1 to 10s; this activity can usually be performed in parallel with actions taken at, for example, the control level. Finally, deliberations at the team level span a conceptually longer period of time (approximately a minute), responding to faults at other levels approximately every 1 to 10s. Crucially, progress at each level is monitored so that task failure (here, task refers to the internal robot tasks undertaken at each level) and excessive backtracking—can be avoided by providing an ability to offload tasks to other levels (or even to a user).

The team level is responsible for decisions involving societal aspects of the robot group. For example, negotiations with other robots on the division of responsibility or the allocation of resources. The team level is also designed to respond to changes in the environment that could impact the performance of the group (e.g., a robot that suddenly detects an intruder entering a team member's sector should realize that if that team member is already tracking another intruder, it will need help). These decisions are guided by processes that are derived from the SharedPlans belief, desire, and intention (BDI) theory of collaboration described below.

The strategy level is concerned with longer-term (in comparison to the control level described shortly) individual decisions involving a robot's intentions (i.e., its commitments to future actions). From a resource-bounded perspective, intentions serve the role of representing fairly stable commitments to actions; central to the strategy level is an ability to reconcile existing intentions with newly considered ones. When a potential intention would conflict with an existing one, the agent must either reject the potential intention or reconsider its existing stock of intentions in the new context, for example, when the potential intention reflects an obstacle that must be overcome or a new opportunity. The strategy level is also responsible for exploring ways in which to achieve an intention, including the means to perform that intention and the resources needed to support execution. Typically, the decisions at this level proceed at a high level of abstraction. For example, in considering a potential intention, strategic-level projection of that action in the context of existing intentions is necessary but generally follows simplified considerations of the factors that will be of relevance at lower levels. An example is navigation around obstacles. Those decisions are handled in a reactive manner. If the obstacle proves impossible to overcome then, the system adapts. By design, control is passed up to the strategy or team level for reconsideration.

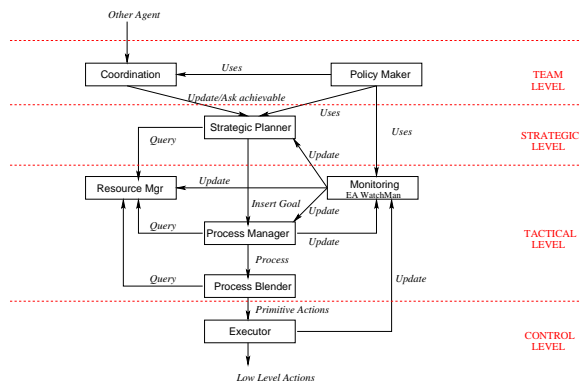


Figure 1: Multilevel architecture

At the tactical level, intended tasks that are the output of the strategy level (along with the expected resources needed for execution) are processed when the time comes for execution. Tasks have associated recipes (each representing a possible means for achieving that task). Each task is matched with a recipe that does not exceed the resources deemed necessary at the strategic level. In addition, monitoring sentinels are attached to the recipe that can be activated during execution for tracking progress and adapting execution to unexpected changes. Adaptation at the tactical level takes the form of the interleaving of multiple tasks that may be intended to be performed at the same time. For example, a robot may wish to follow an intruder while at the same time remaining in communication with a team member. Rather than define a behavior, for every possible combination of behaviors, such as `follow_and_stay_in_communication`, the tactical level implements a scheme for behavior blending (see section 5.2).

The control level is responsible for passing low-level actions for execution to the AV. The control level is also responsible for regularly polling the state of resources on the robot (e.g., battery, camera, motors) and communicating that information to the appropriate level.

From an implementation point of view, the functionalities just described can be realized in a software architecture that is essentially an elaboration on those functionalities. Due to lack of space, we will not go into any further detail regarding the implementation except to say that all the software modules of the current system are implemented in SRI's *Procedural Reasoning System* (PRS) which was developed to serve as a robot control language and is convenient in that it is a BDI system also.

5. TECHNOLOGIES

The SRI robot architecture is based on several years of research at SRI into intelligent reactive control, planning, negotiation, and robot motion control [15, 11, 28, 4, 12]. It is similar to systems like SAFER [10, 25] and SRTA [26] in its ability to deal with multiple goals at once and evaluate when to discard goals. Figure 1 takes a more in-depth look at the Multi-Level Agent Adaptation (MLAA) architecture.

The coordination module receives goal requests from the human commander or other agents, for example to secure a named area of interest and track any unknown vehicles found. The agent participates in a negotiation process to determine its role in achieving the goal, perhaps to patrol

the west sector of the area and initiate a policy to track any evaders. During negotiation, the agent consults the strategic planner to create a plan, or plan segment (referred to as a *recipe*), and assess the recipe's viability given current commitments. If the negotiation process results in the goal and its recipe being accepted, the process manager, the central controller of our execution framework, instantiates the recipe and initiates its execution. For example, a recipe for patrolling a sector might involve a series of steps to search the sector in a lawnmower-type pattern. In addition, it creates monitoring sentinels for use by the EA Watchman to detect deviation from the recipe or policy triggering conditions during execution. The execution of a recipe involves activation of tasks that must be blended with other active tasks to maximize the satisfaction of multiple goals. For example, if the robot needs to reach a waypoint by a set time, take a picture of a location nearby, and also remain concealed, the task blender modifies the path planner at runtime to achieve all three tasks. Finally, the lowest layer in the architecture is the interface between the tasking architecture and the physical, or simulated, robot controller. The fundamental technologies for negotiation, execution and task control, task blending, monitoring and network communication were all developed to overcome the issues of the real physical environments and are described in more detail in the next section.

5.1 Execution and Tasking Control

At the tactical, or execution process control layer there are a number of responsibilities to be met. One or more plans may be instantiated to meet either global, or mission-specific, goals or local policy or self preservation goals. There plans, or recipes, are composed of an ordering of tasks which should be executed to achieve the goal. The issues that arise include:

- How to satisfy or prioritize a set of possibly conflicting or overlapping tasks
- How to react to changes in the sensed world that have stimulated an alert from the EA watchman and may require a change in goals, plans or tasks.
- How to report progress to higher levels of control within the system

The approach taken by this project is to view the problem as an intelligent reactive control system based on the control of processes. It adopts a Belief/Desires and Intentions (BDI) architecture and merges state of the art monitoring and filtering techniques with a novel approach to coordinating multiple behaviors using desirabilities.

Reactive control systems are typically organized around an interpreter that runs a tight control loop of *sensing* to detect key changes in the operating environment or goals, *deliberation* to determine how to respond to sensed changes, and *acting* to execute relevant responses. Our controller is based on the Procedural Reasoning System (PRS), [11]. Here, a highly expressive framework for representing activity called ACT, [15], is used to describe procedures. These procedures are used to describe processes (or plans) to achieve some goal or to serve as appropriate responses to designated events. The tactical process manager is the hub of the autonomous agent, directing its overall operation and behavior. It is responsible for managing the instantiation of plans (committed to

by the strategic, (look ahead), planner), their dynamic decomposition into constituent tasks and behaviors and their coordination with policy and self-preservation behaviors via the motion and sensor control unit on the physical robot platform. Once the set of active, or desirable, behaviors are established their execution is coordinated through a novel approach based on something called *task blending*.

5.2 Task Blending: *coordinating multiple behaviors using desirabilities*

At the task execution level there is a particular problem where several tasks compete for the agent's resources simultaneously. These tasks may be part of one or more higher level goals that the agent is pursuing or may be derived from dynamic responses to unexpected events, such as self-preservation tasks or policy-based tasks.

Conventional approaches to this problem involve simple prioritization of tasks or hard-cored rules for specific conflicting occasions. Instead, we leverage work on context-dependent blending (CDB), [21, 22, 24] from the robotics field of autonomous navigation, to accomplish one task without sacrificing the success of competing tasks.

A significant feature of our approach-distinguishing it from existing conventional treatments- is its reliance on context-dependent comparison of the advantages and disadvantages of alternative control actions from the viewpoint of various reactive and purposive goals. This method avoids a-priori, context-independent, assignment of weights to competing objectives and, unlike conventional approaches based on the determination of a single optimal control option, it chooses a course of action employing an explainable decision rationale. Our method is rooted in a formal logic of preference and utility. It is based on an approach to planning and control in unstructured and uncertain environments, originally developed for the reactive control of autonomous [23]. It exploits logical foundations that regard certain multivalued logics as being related to decision-theoretic notions of relative utility [21] and of similarity between situations [22]. This methodology has been shown in experiments, to lead to robust, flexible, controllers capable of attaining several, possibly conflicting objectives [6].

In a practical solution to the computational complexities of this approach we developed an implementation framework based on the global dynamic window approach to high-speed navigation, [16].

Our general theory for combining lower-level desirability is based on application of fuzzy inference techniques, [9]. The basic idea is that the objective O sought by the supervisory process is the "weighted" conjunction of the individual objectives $O_i, i = 1, ..n$, pursued by the lower-level agents. Context-dependent weighting is modeled by means of state-dependent functions

$$C_i : S \rightarrow [0, 1]$$

, where S is the state space, defining the relevance of each particular objective in the current context, [21]. This weighting corresponds, in effect, to the conjunction of propositions of the form

If the context is C_i , then seek objective $O_i, i = 1, \dots, n$, which are represented in the form

If the context is C_i and the state is s , then the desirability of action u is $D_i(s, u)$,

where u is a control action, and $D_i(s, u)$ is the desirability

function associated with the i -th objective, i.e., a function

$$D_i : SxU \rightarrow [0, 1],$$

where U is the control space.

Since the desirability functions themselves are typically computed employing a set of conditional rules in a rule-set, contextual weighting quantifies, in effect, the extent of applicability of a rule in a given situational context. If the result of the combination of the individual desirability functions $D_i(s, u)$ is denoted $D(s, u)$, then the following expression results from application of fuzzy-logic principles to express the context-dependent conjunction of those preference functions:

$$D(s, u) =$$

$$[C_1(s, u) \Rightarrow D_1(s, u)] \otimes [C_2(s, u) \Rightarrow$$

$$D_2(s, u)] \otimes \dots \otimes [C_n(s, u) \Rightarrow D_n(s, u)]$$

where the operator \otimes represents a triangular (or T-) norm-modeling conjunctions in fuzzy logic and where the symbol \Rightarrow stands for the pseudo-inverse \oslash of \otimes , [2].

In practice, the above expression, is often replaced by its functional, or "disjunctive" counterpart

$$D(s, u) = [C_1(s, u) \otimes D_1(s, u)] \oplus [C_2(s, u) \otimes D_2(s, u)] \oplus \dots \oplus [C_n(s, u) \otimes D_n(s, u)]$$

where the operator \oplus is a triangular, or T-) conorm - modeling disjunction in fuzzy logic.

In many applications, the minimum function is chosen as the T-norm, while the maximum function is employed as the T-conorm. The function $a \otimes b = a.b$, and its associated conorm $a \oplus b = a + b - a.b$ are also employed frequently to model multivalued conjunctions and disjunctions, respectively.

For example, in a single agent scenario an autonomous vehicle may be trying to navigate a route while remaining concealed. Depending on the context the navigate task and conceal task may be decomposed into two competing actions, or behaviors as we will refer to them, move to next waypoint in route vs. remain out of visual range of "the enemy". Similarly, in a multiple agent context shared tasks such as pursue evader or maintain communication connectivity with base may compete with individual goals such as photograph enemy positions in sector A.

It would be nice to reference the whitepaper to say that the specific implementation is described in the white paper.

5.3 Negotiation and collaboration

SRI has been extending and applying this theory for the past several years, together with colleagues at other institutions [7]. In addition, these collaborative processes have been implemented and have formal definitions: for example, "helpful" activity is defined as an activity that reduces the cost that a team member would otherwise incur in executing a task by itself[18]. Negotiations among agents are based on a form of negotiation recently developed at SRI called *structured negotiation* in which negotiation and shared planning activities are interleaved[19]. The fact that each robot is part of a team also means that robots will be honest, trustworthy and place the good of the team above individual

goals; that is, such entities are *not* modeled as the self-interested agents of game theory.

Structured negotiation is a method through which collaborating agents can seek consensus on the apportionment of tasks and resources. The approach draws on research in collaborative planning and human dialog understanding: agent interactions are organized in a manner that reflects the structure of a shared plan. Negotiations are incremental and agent proposals to team members are annotated with causal information that compactly expresses relationships between new proposals and the current context [17]. Normative guidelines for proposal generation further restrict communications of ancillary information to only those fragments that represent departures from the norm. Finally, a set of interpretation rules allows agents to infer information not explicitly communicated.

Structured negotiation embodies the following principles: (1) communications that support negotiation should be efficient, (2) negotiation should be interleaved with planning, (3) processing should be incremental, and (4) interactions should be organized around evolving plans. Roughly speaking, one communication is more efficient than another if its message length is shorter and both communications result in equivalent transmission of information. Such information loading is common in natural language dialogs: when an utterance is interpreted within some context, it will usually carry with it additional information not explicitly transmitted. In bandwidth-restricted environments, efficiency is a desirable property.

The process of collaborative planning is one that takes place over some period of time. It is unrealistic to suppose that agents will suspend negotiations until group deliberations are complete; similarly, it is unrealistic to suppose that agents can suspend deliberations until they have arrived at a consensus regarding the division of tasks and resources. Negotiations must be interleaved with planning; therefore, a communication language for negotiation should be able to refer to elements of a shared plan as well as relations between subplans. When negotiation is interleaved with planning, it cannot range over every possible issue or option at once: this would require that agents negotiate over *every* possible plan; an activity that is computationally prohibitive.

The DARPA ANTS program is developing flexible negotiation strategies for teams of agents. As part of the ANTS program, SRI and Harvard University are developing an incremental, plan-based negotiating technology for teams of autonomous agents. We are applying a theory of collaboration called SharedPlans [7] in which both individual and group plans — either partial or complete — can be represented. We will assume that MICA mission plans are translated into this rich SharedPlans representation.

The SharedPlans theory of collaboration is based on a rich view of plans. Rather than associating a plan for some action α with a group of actions that can achieve α , a plan is instead a structure describing relationships between intentions (commitments) and information needs [20]. Intentions come in two varieties: an intention-to perform some action represents an individual commitment on the part of an agent to perform that action, while an intention-that instead represents a commitment to some condition. Intentions-to serve a number of functions [3]: (1) they constrain deliberations: an agent will seek ways to accomplish an intended action, (2) they represent commitments to action: an agent will

not normally adopt new intentions that conflict with existing ones, and (3) agents monitor the success or failure of attempts to achieve an intention: failures can engender re-planning. Intentions-that play an important role in reasoning about group activities and group plans. Intentions-that also represent commitments, but to certain states or conditions holding. They can engender helpful behavior and also spawn monitoring actions [8].

SharedPlans is formalized in a modal logic. However, the basic elements of a shared plan can be glossed as follows (the items in italics represent the SharedPlans constructs). A group Gr has a shared plan to do α if Gr mutually believe that all members G_i are committed to the following

1. G_i *intends-that* Gr do α .
2. Gr has a (*partial*) task description for α . Plan partiality arises in various ways. For example, the plans distributed by an operator are usually not complete. The strategy manager depends on a certain degree of autonomy on the part of individual robots: each is able to take a skeletal plan from the manager depends on a certain degree of autonomy on the part of individual robots: each is able to take a skeletal plan from the mission planner and execute it in the best way possible. The individual robots, in turn, depend on high-level inputs from the mission planner.
3. *For each constituent act, β , in the task description for α ,*
 - (a) *Some individual or subgroup, G_j , has a (*partial*) plan to do β .*
 - (b) G_j *believes it can do β .* The robots are given partial information by the strategy manager regarding roles of partner robots as well as a local operating context identifying key elements of the environment for acting and monitoring;
 - (c) Gr *mutually believe (a) and (b).* A localized version of this clause is made use of here: robots exchange local operating contexts as execution proceeds in an incremental fashion.
 - (d) G_i *intends-that G_j be able to do β in the context of α .* This is handled through the use of Tachyon sentinels and encapsulate key aspects of plans that a robot should be aware of.
4. Gr has a *full plan to select group member(s) to perform β .* This is implemented through a set of PRSagent-selection procedures in the strategy manager.

5.4 Monitoring

The initial monitoring issues apparent within our domain can be divided into the following four categories:

- Monitoring the completion of, or progress toward, a basic action (e.g., go to a waypoint)
- Monitoring the satisfaction or completion of the multiple tasks to which the robot is currently committed (e.g., pursue evader, patrol area, photograph target every 2 hours)
- Monitoring the activity of unknown or adversarial entities (e.g. evader detected).

- **Monitoring** the state of the communication network, the robot, and other team members (e.g., communication network quality or integrity, robot mobility, or battery level)

For the monitoring in the MLAA, we have adapted the architecture and representations developed for plan and threat monitoring for small unit operations (SUO)[27]. The monitoring is accomplished by two PRS agents, the **Watchman** and the **Manager** (which is part of the MLAA process manager), each running asynchronously. The **Watchman** agent monitors incoming message traffic. It filters irrelevant or insignificantly changed reports, and sends a message to the **Manager** when any report or message requires its attention. The **Manager** agent, after receiving a plan, computes sentinels for the plan and begins plan execution. It compares reports from the **Watchman** agent to the plan and sentinels, and generates high-value alerts without overwhelming the system with too many low-value alerts. The **Manager** continually applies its Acts to respond to new goals and facts posted in its database. The Acts correspond to algorithms for monitoring requirements at each layer in the MLAA architecture. Some implement user alerts and others implement autonomous control.

The inputs to the **Manager** are plans to execute, policy declarations, status reports (including location, speed and orientation) from its own sensor suite, and messages from other agents. These messages include status reports of other agents, reports on mission success or failure, shared information, and requests for help. Depending on communication conditions or policy restrictions, an agent may, or may not, receive from team members status reports (up-to-date locations) of all friendly agents and other entities within visual range. Sentinels are extracted from plans and policy declarations, are evaluated when status reports are received, and may produce alerts. The alerts produced are designed to serve both the autonomous control via the plan manager component, and the user, although the needs of each vary considerably.

Each robot receives two state messages every second from its own sensors and two from each team member, based on network conditions. It also receives similar state messages about entities within its own field of vision. This means in a team of three robots the **Watchman** will be handling a minimum of at least six state messages per second and possibly many more depending on the environment. Our initial monitoring implementation detects the following types of alerts. We plan to implement additional monitoring.

- **At-goal** – robot at current waypoint
- **Stuck** – robot stuck and not at current waypoint
- **Divergent** – robot diverging from current waypoint
- **No-status** – robot no longer reporting its state
- **Target-visible** – robot has a target within its sensor range
- **Lost-target** – robot lost track of target during pursue mission
- **Target-gone** – target moved out of assigned sector during pursue mission

- **Collision** – robot anticipates it will hit a nearby object in the next few seconds
- **Handoff** – robot has delegated/accepted a task to/from another team member

We use the same techniques as the SUO EA for estimating the value of alerts, thus greatly reducing the number of low-value alerts. In particular, we keep event histories for each team member being monitored. These histories are used to determine the value of information and alerts, and to detect **Stuck**, **Divergent**, and **No-status** alerts. The value of issuing an alert takes into consideration latency thresholds and repetition parameters. There are parameters associated with both the automated agent and the user. Some of the agent parameters are customized to improve performance, while others are a function of the behavior of the robot.

An example of how monitoring is used to facilitate autonomous control is illustrated by the situation where an agent is patrolling a designated area. When an evader becomes visible, the agent receives a *Target-visible* alert. Reacting to either a high-priority policy to pursue evaders or an explicit plan step, the agent commits to a new goal *Pursue named-evader*. This goal is achieved by the activation and blending of three tasks: *Follow named-evader*, *Relocate named-evader* and *Search-for named-evader*. Thus, the robot will maintain pursuit even when the evader slips in and out of its field of vision.

The user's preferred strategy might be to report the first sighting of the evader or to track its position, noting whenever it disappears from view. However, the autonomous control requires notification only if the likelihood of recovering visual contact is deteriorating and the robot is searching aimlessly. At this point, a *Target-Lost* alert will be sent to the agent's **Manager** (and possibly the user). In this example, a policy exists for reacting to this type of alert. It will cause the pursuit goal to be dropped and the original *Patrol* plan to be resumed.

No-status alerts have proven useful to the human user, as they indicate a hardware or software problem on a robot or the network. Monitoring allows such problems to be recognized immediately. These problems took considerably longer to detect and diagnose before monitoring was implemented. *At-goal*, *Stuck*, and *Divergent* are essential alerts for the autonomous-control agent-navigation system, as well as being useful to a human user who wants to closely monitor the progress of a robot. Subtleties of the domain must be considered to avoid false alarms. For example, the robot may be paused because of GPS uncertainty and the GPS should be given time to establish connection with satellites. Also, a robot takes time to turn and thus should not be regarded as stuck or divergent until turns and steering adjustments have had time to complete.

Target-visible, *Target-lost*, and *Handoff* are useful to both the user and the autonomous controller, particularly when the task is to monitor or pursue a target. The autonomous controller requires immediate awareness of loss of sensor contact, so it can adjust its lower-level behavior or sensor parameters to find the evader. However, such immediate alerts would be unproductive for the human user or the plan-level controller, and alerting is controlled by a customizable interval. This interval gives the agent time to relocate the evader, possibly avoiding an alert. These types of alerts are the most time critical in our evaluation domain.

5.5 Communications

5.5.1 Mobile Ad-Hoc Networking

Communications among the robots, between the robots and the simulator, and between the GPS base station and GPS receivers are done through wireless networking. The underlying protocol is called Topology Broadcast based on Reverse Path Forwarding (TBRPF) [1], which is a link state routing protocol designed for mobile ad-hoc networks. This protocol allows robust dynamic rerouting of communications through all connected nodes in the network. In practice, this means that when two sets of robots become disconnected, they can be reconnected by assigning one or more robots to act as relays between the two sets. Such relay connections are made automatically once communication between the relay and the two sets of robots becomes available. It is important to note that the protocol itself handles establishing any relay connection—there are no explicitly assigned router nodes in the network. Relays may be used to extend communications over a distance that would not normally be surmountable with ordinary wireless networking. They may also be used in cases where the environment prevents line of sight communications. An example of the latter is a situation in which many buildings or even small RF (radio frequency) blocking obstacles surround and obstruct the team of robots. Having this protocol allows us to make the more reasonable assumption (compared with the simplifying assumption in section 2) that team level actions may be taken to ensure that communications between robots can be established, when the need arises. In fact, with sufficient saturation of the environment with robots, this protocol almost allows us to make the simplifying communication assumption. For most environments, this saturation is achieved when, for any distinct robots r_s and r_e , there is some sequence of robots, r_0, r_1, \dots, r_n such that $r_0 = r_s, r_n = r_e$ and there is a line of sight path between any two robots r_i, r_{i+1} . Thus, for open fields, we need relatively few robots to achieve saturation, and even for corridor type environments, one generally needs only one robot per intersection of two or more corridors.

5.5.2 OAA

On top of the network protocol, the various high level software components in the system—the robot control, the simulator and the interface—communicate using the Open Agent Architecture (OAA)[14, 5] in asynchronous mode. Each robot has access to an OAA facilitator that provides local communications. Data that needs to be sent outside one of the software components uses an OAA agent that transmits data to other OAA facilitators on other robots, allowing for greater decentralization than the OAA protocol, alone, permits. Using this agent allows us to change from OAA's default transport protocol, TCP, to some other protocol, or even a number of protocols, designed to handle large latencies. Assuming that TCP communications do not time out, it is even possible to eliminate the inter-facilitator module and use a single facilitator for all communications between robots and between robots and the simulator.

OAA does slow down communications in the sense that the number of messages that we could send is slower compared to the situation we would have if we had limited the implementation to remote function calls. However, the components in our system are not all written in the same lan-

guage. Some are written in C++, some in Java and some in Lisp. Using OAA allows these components to communicate without the requirement to translate all components to a common language.

5.6 Open system experimental framework and low-level control

5.7 Hardware

To test our design, we have 3 Pioneer AT robots from ActivMedia. Each robot runs Linux on an AMD K6 or Intel Pentium III processor. Localization is accomplished with a Novatel OEM4 GPS (global positioning system) receiver and antenna, Crossbow INS (inertial navigation system), and digital compass. An 802.11b compliant Orinoco wireless PCMCIA card and antenna provides communications between robots as well as to the simulator (see section 5.8.1). Low level obstacle avoidance; that is, obstacle avoidance that is not handled by higher level path planning, is accomplished using sonar detectors on the robots.

Other sensors, such as the sensing of other nearby robots is done using a simulation of such sensors. Given the GPS coordinates of all robots and a map of the region around the robot, we can simulate many different types of sensors. Currently, we model sensors that ignores obstacles, and which have a limited radius of effectiveness which differs between UGVs and UAVs.

5.8 Software

5.8.1 Simulator

The SRI Augmented Reality Simulator (SARS) is a new multi-agent simulator specifically designed to simulate robots and UAVs. SARS models the world for all agents involved in the simulation. It produces the same output in terms of sensors, actuators, and resources (battery status, communications range). SARS computation and simulation is based on a very precise 3D model of the environment. SARS is precise enough that we can mix real physical robots moving in the real world with virtual evaders and see the physical robots following a virtual evader. The notion of Augmented Reality came from this feature. Using SARS we are able to simulate a team of UAVs flying in formation on a larger battlefield than we actually have access to. The team of UAVs may be larger than what we physically have, as well. Our research about teams of UAVS or robots could not be realistically tested on physical robots due to high cost of maintaining a small fleet. SARS was one way for us to test large and more complex scenarios without sacrificing too much realism.

5.8.2 Robot Control

The UGVs are controlled using Saphira [13]. Saphira allows for both low level control, as well as low level behaviour. Examples of such behaviour include actions like avoiding obstacles that did not appear in the execution and task control, or avoiding obstacles that are not in the same position as higher reasoning modules initially thought, or maintaining a straight line, or maintaining a small distance from walls in a corridor. At this level of software control, the robot is responsible for sending out sensor readings in which higher reasoning modules might be interested. Currently, this is only the current GPS position and current velocity,

but might include things such as battery levels, sonar readings, and even raw sensor readings (such as might come from the motor actuators, or the sonar).

Each robot has all the software it needs to allow it to interact with its team members. This includes all the communications software, described in section 5.5, the execution and task control software (section 5.1) as well as the robot control software itself.

6. VIDEO

A video of this working system is available from our web site at <http://www.ai.sri.com/movies/UCAV-simivalley-demo-2002-small.mov>. In this video, you can see the actual demonstration of the scenario describes in the section 3. The robots are Pioneer 2 AT equipped with INS, compass, GPS, P-III 800Mhz on-board computer. The evader is 1/4 scale radio-controlled car controlled by me. The robots are completely stand-alone and autonomous.

7. SUMMARY

8. REFERENCES

- [1] B. Bellur and R. Ogier. A reliable, efficient topology broadcast protocol for dynamic networks. In *Proceedings of IEEE INFOCOM*, March 1999.
- [2] D. Boixader and L. Godo. Handbook of fuzzy computation. chapter Fuzzy inference, pages 5.1–5.28. Institute of Physics Publishing, 1998.
- [3] M. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.
- [4] A. Cheyer and D. Martin. The open agent architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1):143–148, 2001.
- [5] A. Cheyer and D. Martin. The open agent architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1):143–148, March 2001. OAA.
- [6] C. Congdon, M. Huber, D. Kortenkamp, K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti. Carmel vs flakey: A comparison of two robots. *AI Magazine*, 1(4):49–578, 1993.
- [7] B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(1):269–357, 1996.
- [8] B. J. Grosz and S. Kraus. The evolution of SharedPlans. In A. Rao and M. Wooldridge, editors, *Foundations and Theories of Rational Agency*, 1998.
- [9] H. Hellendoorn and C. Thomas. Defuzzification in fuzzy controllers. *Jour. of Intelligent and Fuzzy Systems*, 1:109–123, 1993.
- [10] G. Holness, D. Karuppiah, S. Uppala, and S. C. Ravela. A service paradigm for reconfigurable agents. In *Proc. of the 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS (Agents 2001)*, Montreal, 2001.
- [11] K.L.Myers. A procedural knowledge approach to task-level control. In *Proceedings of the Third International Conference on AI Planning Systems*, 1996.
- [12] K. Konolige and K. Myers. *Artificial Intelligence Based Mobile Robots: Case studies of Successful Robot Systems*, chapter The Saphira architecture: a design for autonomy. MIT Press, 1998.
- [13] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti. The saphira architecture: a design for autonomy. *Journal of Experimental and Theoretical AI*, 1996.
- [14] D. L. Martin, A. J. Cheyer, and D. B. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):91–128, January-March 1999. OAA.
- [15] K. Myers and D. Wilkens. The act editor user's guide. Technical report, Artificial Intelligence Center, SRI International, 1997.
- [16] O. B. O and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Hawaii, USA, 1999.
- [17] C. L. Ortiz. A commonsense language for reasoning about causation and rational action. *Artificial Intelligence*, 111:73–130, 1999.
- [18] C. L. Ortiz. Introspective and elaborative processes in rational agents. *Annals of Mathematics and Artificial Intelligence*, 25(1–2):1–34, 1999.
- [19] C. L. Ortiz and E. Hsu. Structured negotiation. In *Proceedings of the First International Conference on Autonomous agents and multiagent systems*, 2002.
- [20] M. Pollack. *Plans as Complex Mental Attitudes*, pages 77–103. MIT Press, 1990.
- [21] E. Ruspini. Truth as utility: A conceptual synthesis. In *Proceedings of the 7th Conf. On Uncertainty in Artificial Intelligence*, pages 316–322, LA, CA, 1991.
- [22] E. Ruspini. *Fuzzy Logic and Neural Network Handbook*, chapter On the semantics of approximate reasoning, pages 5.1–5.28. McGraw Hill, 1996.
- [23] A. Saffiotti, K. Konolige, and E. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2):481–526, 1995.
- [24] A. Saffiotti, E. Ruspini, and K. Konolige. Blending reactivity and goal-directedness in a fuzzy controlled. In *Proceedings of the IEEE Int. Conf. On Fuzzy Systems*, pages 134–139, CA, 1993. IEEE Press.
- [25] J. Sweeney, T. Brunette, Y. Yang, and R. Grupen. Coordinated teams of reactive mobile platforms. In *Proceedings of the 2002 IEEE Conference on Robotics and Automation*, Washington, D.C., 2002. IEEE.
- [26] R. Vincent, B. Horling, V. Lesser, and T. Wagner. Implementing soft real-time agent control. In *Proceedings of the 5th International Conference on Autonomous Agents*. ACM Press, 2001.
- [27] D. E. Wilkins and T. J. Lee. Action-specific execution monitoring for small unit operations. Final Report SRI Contract 7150, Artificial Intelligence Center, Menlo Park, CA, August 2001.
- [28] D. E. Wilkins and K. L. Myers. A multiagent planning architecture. In *Proceedings of the Fourth International Conference on AI Planning Systems*, 1998.