Shared Activity Coordination

Bradley J. Clement and Anthony C. Barrett
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, M/S 126-347, Pasadena, CA 91109-8099 USA
voice +1-818-393-4729, fax +1-818-393-5244
{bclement, barrett}@aig.jpl.nasa.gov
Keywords: multiagent systems, planning, scheduling, real-time, communication

Abstract

While multiagent planning research has largely concentrated on distributing a planning problem or resolving conflicts among collaborative agents' plans, it has focused less on communication constraints, real-time issues, and negotation of self-interested agents. In domains where agents that interleave planning and execution have varying degrees of interaction and different constraints on communication and computation, agents will require different coordination protocols in order to efficiently reach consensus in real time. We briefly describe a largely unexplored class of real-time, distributed planning problems (inspired by interacting spacecraft missions), new challenges they pose, and a general approach to solving the problems. These problems involve self-interested agents that have infrequent communication but coordinate over joint activities and shared resources. We describe a Shared Activity Coordination (SHAC) framework that provides a decentralized algorithm for negotiating the scheduling of shared activities over the lifetimes of multiple agents, a soft real-time approach to reaching consensus during execution with limited communication, and a foundation for customizing protocols for negotiating planner interactions. We apply SHAC to a realistic simulation of interacting Mars spacecraft and illustrate the simplicity of protocol development.

1 Introduction

Interacting agents that interleave planning and execution must reach consensus on their commitments to each other before executing interdependent activities. When interleaving planning and execution, an agent adjusts its planned activities as it gathers information about the environment and encounters unexpected events. Interacting agents coordinate these adjustments to manage commitments with each other, but in the presence of communication constraints, reaching consensus on these commitments must be planned to avoid inconsistent execution. Demand for this kind of autonomous agent technology is growing for space applications. Autonomous spacecraft promise new capabilities and cost improvements in exploring the solar system. Spacecraft (and rovers) that explore other planets have intermittent, delayed communication with Earth, requiring that they be able to manage their resources and operate for long periods in isolation. The common approach to autonomous decision making is to place integrated data analysis, planning, and execution systems onboard the spacecraft.

In addition, there is a growing trend toward multi-spacecraft missions. Over forty multi-spacecraft missions have been proposed, including formation flying teams and over 16 planned missions to Mars in the next decade. These spacecraft will coordinate measurements, share data, and route data to and form Earth. Separate missions, such as those to Mars have their own budgets, experiments, and operations teams. As such, the spacecraft represent self-interested entities that benefit from collaborative interactions.

But, even a single spacecraft has multiple science instruments for executing different goals of different scientists, and different operations groups will have different areas of expertise over different subsystems for control. These different groups negotiate over mission plans in the same way that different Mars missions must collaborate over spacecraft interactions. Whether this negotiation is done on-board or on Earth, there is a distributed operations planning problem that benefits from automation. Both also have real-time aspects. On-board systems must plan safely over near- and long-term horizons, and ground systems must also converge on planss for daily, weekly, and lifelong mission operations. Ground planning also suffers from communication constraints. Scientists from different universities or opposite sides of the globe will intermittently provide inputs and respond on an irregular basis. A collaboration/negotiation system must be built around communication constraints to meet hard deadlines for coming to consensus on consistent operations plans.

In this work, we will briefly characterize this general problem in terms of activity interaction types and communication constraints and discuss its challenges. The field of multiagent planning has largely focused on fully cooperative planning and execution [Decker, 1995; desJardins and

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Wolverton, 1999; Tambe, 1997; Grosz and Kraus, 1996; Clement and Durfee, 2000]. Market-based agent systems address near-term resource negotiation but have rarely addressed how near-term decisions affect longer-term goals. Multiagent systems built for Robocup Soccer competitions mainly address collaborative multiagent execution in an adversarial environment and have limited planning capabilities. These approaches do not adequately address real-time planning for self-interested agents.

We also present a framework for Shared Activity Coordination (SHAC). SHAC consists of an algorithm for continually coordinating agents and a foundation for rapidly designing and implementing coordination protocols based on a model of shared activities. In the same fashion that a real-time planning system must commit to actions to pass to an execution system, a real-time coordination system must additionally establish consensus on shared activities before they are executed based on communication constraints. Our ultimate goal is to create interacting agents that autonomously adjust their coordination protocols with respect to unexpected events and changes in communication or computation constraints so that the agents can most efficiently achieve their goals.

First we characterize a class of real-time, self-interested multiagent planning problems. Then we describe the shared activity model, the SHAC algorithm, and its interface to the planner. We also specify some generic roles and protocols using the SHAC framework that build on prior coordination mechanisms. In addition, we present an algorithm for determining how long a protocol will reach consensus under particular communication restrictions. Then we describe how our implementation of SHAC currently is used to coordinate the communication of two rovers and three orbiters in a simulated Mars scenario. We follow with future research needs revealed in this scenario and comparisons to related work.

2 Continual Coordination Problem

As mentioned before, agents that interleave planning and execution must commit near-term activities to the execution system while receiving feedback in the form of state updates and activity performance. One such continual planning system, CASPER (Continuous Activity Scheduling Planning Execution and Replanning) identifies the period when the planner commits activities to the execution system as a *commit window* [Chien *et al.*, 2000]. While the planner must resolve conflicts on activities before they are committed to execution, distributed planning agents must additionally reach consensus on team interactions before execution. As explored in the team plan model given by TEAMCORE [Tambe, 1997; Pynadath *et al.*, 1999], formalizations of Shared Plans [Grosz and Kraus, 1996], and coordination interactions of TAEMS [Decker, 1995], these interactions could include

- use and replenishment of shared resources,
- joint actions for achieving a mutually beneficial subgoal,
- choice of methods for achieving a team subgoal,
- participation and role assignments in a joint plan, and
- proposals and commitments of the above.

However, reaching consensus on these interactions is complicated when the agents can only communicate intermittently. Depending on the number of agents involved in a particular interaction, a consensus protocol may need to be initiated far in advance and negotiations settled far in advance of execution. Thus, for any particular set of interactions, a *consensus window*, within which the agents must limit negotiation and establish agreement, should be defined. For example, if three agents must negotiate a joint action in advance of execution but can only communicate pairwise in disjoint time windows, a consensus window must extend at least to cover windows connecting all three agents. Inside the consensus window, a simple protocol eliminating negotiation (such as all agree or reject) must be employed to guarantee consensus. Interactions beyond the consensus window can be negotiated with more elaborate protocols.

We informally describe the continual coordination problem because our approach is intended to be general to the capabilities of the individual planning and execution systems. The continual coordination problem can be specified generally as a continual planning domain and problem instance for each agent as well as communication constraints between agents. Continual planning problems have an evolving set of goals (as opposed to a single goal) and performance is measured as a function of the goals successfully achieved (executed) within a time horizon. The multiagent aspect of the problem is that interactions (such as those listed previously) create dependencies between the planning and execution systems, over which the agents must coordinate. The communication constraints can be time windows within which subsets of agents can communicate, the reliability of communication, the costs of communication (e.g. privacy), the bandwidth of channels, *etc.* These constraints determine how the agents can achieve consensus for a particular communications protocol.

Thus, many decisions must be made in the design of an efficient continual coordination system. What planning and execution systems are appropriate? What protocols should be used for negotiation and consensus? How should the consensus window be specified? Instead of providing a general solution to all of these questions, we describe an implemented framework for designing and evaluating protocols to negotiate interactions and establish consensus.

3 SHAC

Our approach, called Shared Activity Coordination (SHAC), provides a general algorithm for interleaving planning and the exchange of plan information based on shared activities. Agents coordinate their plans by establishing consensus on the parameters of shared activities. Figure 1 illustrates this approach where three agents share one activity and two share another. The constraints denote equality requirements between shared activity parameters in different agents. The left vertical box over each planner's schedule represents a *commit window* that moves along with the current time. A consensus window is shown to the right of the commit window, within which consensus must be quickly established before committing. Since consensus is hard to maintain when all agents can modify a shared activity's parameters at the same time, agents must participate in different coordination roles that specify which agent has control of the activity. As shown

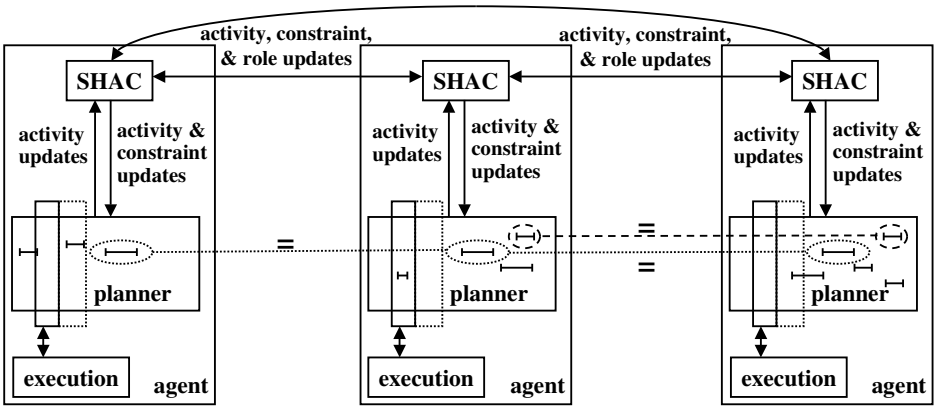

Figure 1: Shared activity coordination

in the figure, SHAC interacts with the planning and execution by propagating changes to the activities, including their parameters and constraints on the values of those parameters.

SHAC continually coordinates by interfacing to a combined planning/execution system that responds to failures and state updates from the execution system. Our implementation interfaces with one such continual planning system, CASPER, mentioned in the previous section. Instead of batch-planning in episodes, CASPER continually adapts near and long-term activities while re-projecting state and resource profiles based on updates from sensors.

3.1 Shared Activities

The model of a shared activity is meant to capture the information that agents must share, including control mechanisms for changing that information. A shared activity is a tuple $\langle parameters, agent\ roles, protocols, decomposition, constraints \rangle$. The parameters are the shared variables and current values over which agents must reach consensus by the time the activity executes. The agent roles determine the local activity of each agent corresponding to the joint action. To provide flexible coordination relationships, the role activities of the shared activity can have different conditions and effects as specified by the local planning model. The shared parameters map to local parameters in the role activity.

For example, as shown by the shared activity instance in Figure 2, a data communication activity can map one agent to a `receive` role activity and another to a `send` role activity. Shared parameters could specify the start time, duration, transfer rate, and data size of the activity. The data size is depleted from the sender's memory resource but added to the receiver's memory. The agents could have separate power usages for transmitting and receiving. In this case the resources are not shared. Another shared activity could be the use of a common transport resource. Although one agent in an active transit role actually changes position, other agents in passive roles have local activities that only reserve the transport resource.

Protocols are mechanisms assigned to each agent (or role) that allow them to change constraints on a shared activity, the set of agents assigned to the activity, and their roles. In Figure 2, both agents use an argumentation protocol to negotiate the scheduling and attributes of the communication.

The shared decomposition enables agents to select different team methods for accomplishing a higher level shared goal. Specifically, the decomposition is a set of shared subactivities. The agents can choose the decomposition from a pre-specified set of subactivity lists. For example, a joint observation among orbiters could decompose into either (`measure`, `process_image`, `downlink`) or (`measure`, `downlink`).

3.2 Constraints

Constraints are created by agents' protocols to restrict sets of values for parameters (*parameter constraints*) and permissions for manipulating the parameters, changing constraints on the parameters, and scheduling shared activities (*permission constraints*). These constraints restrict the privileges (or responsibilities) of agents in making coordinated planning decisions. By communicating constraints, protocols can come to agreement on the scheduling of an activity without sharing all details of their local plans.

A parameter constraint is a tuple $\langle agent, parameter, value\ set \rangle$. The $agent$ denotes who created the constraint. Some protocols differentiate their treatment of constraints based on the agent that created them. For example, the asynchronous weak commitment algorithm prioritizes agents so that lower-priority agents only conform to higher-priority agent constraints [Yokoo and Hirayama, 1998]. Agents can add to their constraints on a parameter, replace constraints, or cancel them. A string parameter constraint, for example, can restrict a parameter to a specific set of strings. An integer or floating point variable constraint is a set of disjoint ranges of numbers. Scheduling constraints can be represented as constraints on a start time integer parameter. This is shown in Figure 2 where the rover restricts the start time of the communication between two eight minute intervals.

Permission constraints determine how an agent's planner is allowed to manipulate shared activities. The following permissions are currently defined for SHAC:

- parameters - change parameter values
- move - set start time

```
shared_activity communicate comm_id_12 {
  time start_time = 2004-302:09:30:00; // date
  int duration =  200;    // seconds
  int data_size = 25600;  // 25.6 Mbits
  real xmit_rate = 128.0; // Kbps
  int priority = 1;       // critical
  roles =
    receive by orbiter,
    send by rover;
  protocols =
    receive argumentation,
    send argumentation;
  permissions =
    receive (move, delete, xmit_rate),
    send (delete, data_size, priority);
  parameter_constraints =
    rover start_time = ([2004-302:09:30:00, 2004-302:09:38:00],
                        [2004-302:18:30:00, 2004-302:18:38:00]);
}
```

Figure 2: An instance of a shared communication activity between a rover and orbiter

- duration - change duration of task
- delete - remove from plan
- choose decomposition - select shared subactivity of an *or* activity
- add - add to plan[1]
- constrain - send constraints to other agents

In the communication example in Figure 2, the receiver is allowed to reschedule (move) the activity, delete it, or change the transmission rate. The sender cannot move the activity, but can delete it and change the requested size and priority.

3.3 Coordination Algorithm

The SHAC algorithm negotiates the scheduling and parameter values of shared activities until consensus is reached. Figure 3 gives a general specification of the algorithm. SHAC is implemented independently of the planner. Steps 1 through 3 are handled by the planner through an interface to SHAC. Step 4 invokes the protocols that potentially make changes to refocus coordination on resolving shared activity conflicts and improving plan utility. SHAC sends modifications of shared activities and constraints to sharing agents in step 5. In step 6, shared activities and constraints are updated based on changes received from other agents.

Ignoring coordination, a continuous planner must determine when it is appropriate to release activities to the execution system. In some cases, an activity involved in a conflict may either be released (requiring the planner to recover from potential failures) or postponed (to allow the planner to recover before a failure occurs). CASPER keeps a *commit window* (an interval between the current time and some point in the near future) within which activities cannot be modified and passes these activities to the execution system.

[1]This permission applies to a class of shared activities (i.e. an agent may be permitted to instantiate a shared activity of a particular class).

This interaction with the executive becomes more complicated when agents share tasks. SHAC must ensure that, when a shared activity is released, all agents release it while in consensus on the start time and other parameters of the task. Ideally the agents should establish consensus before the commit window. SHAC avoids changes in the commit window by keeping a *consensus window* that extends from the commit window forward by some period specific for the activity. As time moves forward, the windows extend forward. When a shared activity moves into the consensus window, the agents switch to the simple consensus protocol to try and reach consensus before the activity moves into the commit window.

4 Protocols

In general, protocols determine when to communicate, what to communicate, and how to process received communication. During each iteration of the loop of the coordination algorithm (Figure 3), the protocol determines what to communicate and how to process communication. A protocol is defined by how it implements the following procedures to be called during step 4 of the SHAC coordination algorithm for the shared activity to which it is assigned:

1. modify permissions of the sharing agents
2. modify locally generated parameter constraints
3. add/delete agents sharing the activity
4. change roles of sharing agents

The default protocol, representing a base class from which other protocols inherit, does nothing for these methods. However, even with this passive protocol, the SHAC algorithm still provides several capabilities:

joint intention A shared activity by itself represents a joint intention among the agents that share it.

mutual belief Parameters or state assertions of shared activities can be updated by sharing agents to establish consensus over shared information.

Given: a $plan$ with multiple activities including a set of $shared\_activities$ with $constraints$ and a $projection$ of $plan$ into the future.

1. Revise $projection$ using the currently perceived state and any newly added goal activities.
2. Alter $plan$ and $projection$ while honoring $constraints$.
3. Release relevant near-term activities of $plan$ to the real-time execution system.
4. For each shared activity in $shared\_activities$,
   - if outside consensus window,
     - apply each associated protocol to modify the shared activity;
   - else
     - apply simple consensus protocol.
5. Communicate changes in $shared\_activities$.
6. Update $shared\_activities$ based on received communications.
7. Go to 1.

Figure 3: Shared activity coordination algorithm

resource sharing Sharing agents can have identical constraints on shared states or resources.

active/passive roles Some sharing agents can have active roles with execution primitives while others have passive roles without execution primitives.

master/slave roles A master agent can have permission to schedule/modify an activity that a slave (which has no permissions) must plan around.

The following sections describe some subclasses of this abstract protocol, demonstrating capabilities that each protocol method can provide.

4.1 Argumentation

Argumentation is a technique for negotiating joint beliefs or intentions [Kraus *et al.*, 1998]. Commonly, one agent makes a proposal to others with justifications. The others evaluate the argument and either accept it or counter-propose with added justifications. This technique has been applied to teamwork negotiation to form teams, reorganize teams, and resolve conflicts over members' beliefs [Tambe and Jung, 1999]. It can also be used to establish consensus on shared activities.

A shared activity and associated parameter values are the proposal or counterproposal. Justifications are given as parameter constraints. A proposal is a change to a shared activity that does not violate any parameter constraints. A counterproposal may violate constraints. Protocol method 2 must be implemented to provide the parameter constraint justifications for proposals and counter-proposals. In order to avoid race conditions, protocol method 1 regulates permissions.

Argumentation method 1

- if this agent sent the most recent proposal/counterproposal
  - if planner modified shared activity
    - remove self's modification permissions
- else
  - give self modification permissions (e.g. move and delete)

Argumentation method 2

- if planner modified shared activity
  - generate parameter constraints describing locally consistent values

For example, one agent can propose an activity with a particular start time and add justifications in the form of all intervals within which the shared activity can be locally scheduled. Other agents can replan to accommodate the proposal and counter-propose with their own interval restrictions if replanning cannot accommodate others' constraints. If the agents cannot establish consensus before the consensus window, a higher ranking agent can mandate a time that benefits most of the agents. Of course, there are many variations on this example. Agents may be restricted because they are slaves or do not have constraint permissions to counter-propose.

4.2 Delegation

Delegation is a mechanism where an agent in a passive delegator role assigns and reassigns activities to different subsets of agents in active subordinate roles. The delegator and subordinate protocols only need to implement protocol method 3 to choose the subordinates sharing the activity.

Delegator method 3

- if $agent\ roles$ empty
  - choose an $agent$ to whom to delegate the activity
  - add ($agent$, subordinate) to $agent\ roles$

Subordinate method 3

- if cannot resolve conflicts/threats involving activity
  - remove self from $agent\ roles$

5 Defining Consensus Windows

Here we describe an algorithm for determining the shortest consensus window given a consensus protocol under particular communication constraints. Suppose agents have predetermined communication opportunity windows, such as orbiters having regular patterns of visibility to each other, a planetary surface, or Earth. We assume that communication is reliable with delivery time guarantees, that bandwidth is sufficient for coordination protocol communications, and that

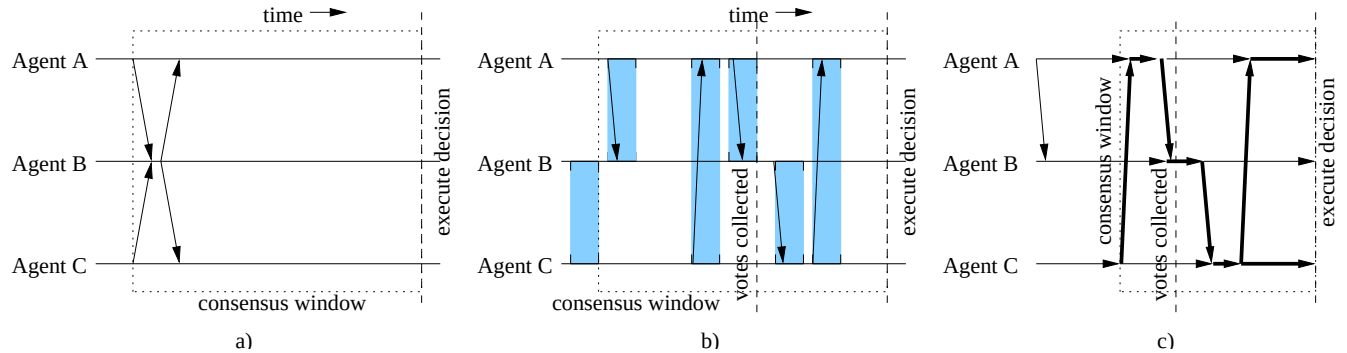

Figure 5: State-time diagrams of agent communication. a) no communication constraints; b) communication restricted to windows of opportunity; c) directed acyclic graph of information flow

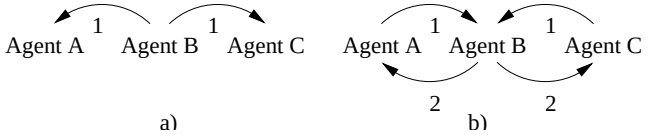

Figure 4: Information flow for consensus protocols: a) highest rank decides and b) voting or auction

the planner has worked out contention for power and memory resources required for communication.

Consensus protocols can be classified according to the flow of data and the computation time required to respond. Figure 4 shows examples for highest-rank-wins, voting, and auction protocols.[2] The voting protocol requires each participating agent to send a vote to a central agent, who (once all votes are received) can instantaneously determine the outcome that it must send to all participating agents. The space-time diagram in Figure 5a shows how voting reaches consensus for a decision when triggered upon entering the consensus window. However, this diagram assumes that agents can communicate at any time. Figure 5b shows the same protocol when pairs of agents can only communicate during specified periods (indicated by shaded regions). By walking through this diagram from left to right, the steps of the information flow diagram of Figure 4 are realized. In order to minimize the time to consensus, Agent C sends its vote/bid to B through A, and B sends the vote/auction outcome to A through C.

In order to determine how long the consensus window should be, we expand backwards through the state-time diagram according to the information flow diagram. To do this, we first construct a directed graph (G_c) as shown in Figure 5c from the execution time backwards, where each node is labeled by an estimated time value and an agent identifier. We also construct another directed graph (G_i) for the protocol's information flow, labeling edges according the state ordering (as done in Figure 4). Using these two graphs, the following algorithm determines the time to begin the consensus window by expanding a frontier of vertices backwards for each step (state) of the information flow diagram.

[2]We assume that the decision being voted or auctioned is priorly known by the agents.

Function Determine start time of consensus window
Input directed acyclic graph of communication opportunities $G_c(V_c, E_c)$, information flow diagram $G_i(V_i, E_i)$
Output $windowStart$

- $windowStart$ = execution time (i.e. $\max_{v \in V_c}(v.time)$))
- $state$ = $\max_{e \in E_i}(e.state)$
- repeat
  - $flow$ = $\{e \in E_i | e.state = state\}$
  - $synchTime$ = $windowStart$
  - for each $e_i \in flow$
    - $frontier$ = $\{v \in V_c | v.agent = e_i.dest \wedge v.time = synchTime\}$
    - repeat
      - E = $\{e \in V_c | e.dest \in frontier \wedge e.source \notin frontier\}$
      - e_c = $\text{argmin}_{e \in E}(e.source.time)$
      - $frontier$ = $frontier$ + $\{e_c.source\}$
    - until $e_c.source.agent = e_i.dest$
    - $windowStart$ = min($windowStart$, $e_c.source.time$)
  - $state$ = $state$ - 1
- until $state$ = 0
- return $windowStart$

The algorithm begins by setting $windowStart$ to the time of execution. It works backwards through the states of the information flow diagram, so $state$ is initially set to 2 for the voting protocol. The $flow$ set contains all of the edges in the flow diagram labeled 2. $synchTime$ is the time point between states and is used to populate $frontier$ with a vertice for each destination agent (agents A and C for the voting protocol) in the communication opportunity graph (G_c) at $synchTime$. After working back to state 1, $frontier$ only contains the vertice for Agent B just after the time that all votes are collected. The $frontier$ is greedily expanded backwards in the fashion of a tree spanning algorithm, iteratively adding vertices at time points closest to the frontier. Once the source agent of the information flow edge is reached, $windowStart$ records that latest time the source agent can send out information. This is done for each edge of the $flow$ for the same state since it can take longer for information to travel from different agents. Once all edges of one state are simulated, $windowStart$ contains the minimum time, representing the synchronization point ($synchTime$) for the prior

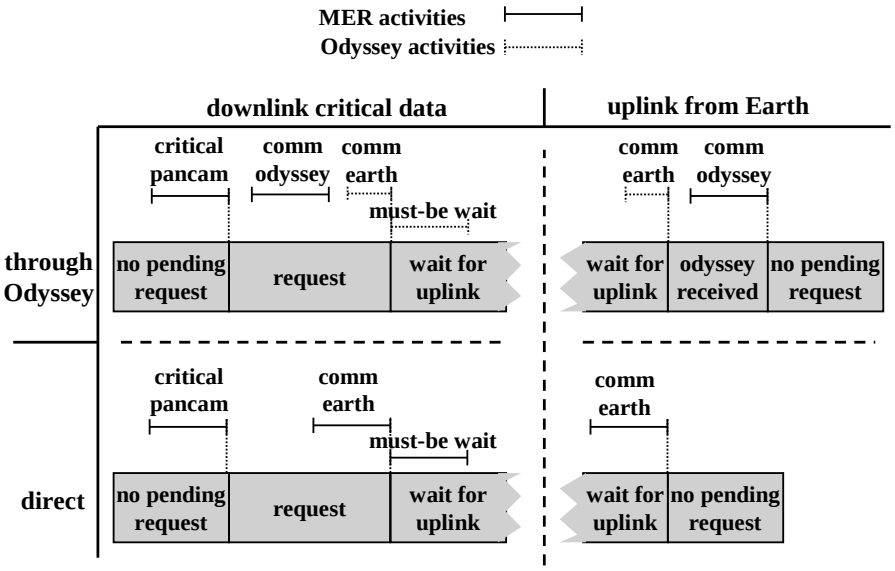

Figure 6: Downlink/uplink states for a rover

state to be simulated. Figure 5c shows thicker edges used to expand the frontiers of each state to find the latest possible consensus window start for the voting protocol. This is the actual flow of information for reaching consensus. Notice that the optimal start time is later than the example in Figure 5b.

The algorithm assumes that communication constraints are windows of communication and bounded delays on data transfer. The algorithm also assumes a simplified version of the information flow diagram that models the protocols. A possible expansion could include probabilistic information for unreliable transfer of data, and the algorithm could be adapted to give consensus windows with a probability of reaching consenus. Communications costs could also be included so that the algorithm could return optional consensus windows with different overall costs. While parallel information flow is modeled, the algorithm does not handle multiple hops within a single state. While somewhat limited, this approach can serve as a starting point for incorporating these extended capabilities.

6 Application to Mars Scenario

Now we describe how SHAC is applied to a simulated three-day scenario involving two Mars Exploration Rovers (MERs), the Mars Odyssey orbiter, Mars Global Surveyor, and the Mars Express orbiter. The delegation protocol described previously was subclassed for the rovers to assign and reassign the routing of images to the orbiters based on how quickly they can deliver the data to Earth. Different master/slave and active/passive roles are defined using permission constraints for the shared activities to implement a basic protocol for coordinating communication to and from Earth. Interactions over communication (once delegated) are between two agents, so the consensus window is defined to cover communication activities spanning two communication opportunities into the future. Once in the consensus window, the rover cannot redelegate activities unless the orbiter cannot resolve conflicts and must decommit. We intend to experiment with other protocols and consensus window definitions in this domain in our future work.

The MERs (MER A and MER B) and the orbiters can communicate with Earth directly, but the MERs can optionally route data through the orbiters, which talk with Earth at a higher bandwidth. The rovers need daily communication with ground operations to receive new goals. The rovers will often fail to traverse to a new target location and cannot proceed until new instructions come from ground operations. In this scenario both MERs must negotiate with the assigned orbiter to determine how to most quickly get a response from Earth after sending an image of their surroundings.

Each MER has a communication state shared with each orbiter that tracks when the image is generated, when it gets to Earth, and when the response from ground operations arrives to the rover. Shared activities for changing the state are shown for different routing options in Figure 6. The rover's activity for generating an image from its panoramic camera changes the state to `request` to communicate its need to downlink and receive an uplink. Activities for sending the image to Earth (either directly or through Odyssey) change the state to `wait for uplink` to indicate that the rover will then be waiting for the uplink. Ground operations needs a period of time to generate new commands for the uplink, so if the uplink is received by Odyssey, the state changes to `received` to indicate that now the rover can get the uplink from Odyssey. Once the rover receives the uplink, the state changes back to the normal `no pending request` state. Rover tasks (such as a traverse) need the uplinked data before executing, so it places a local constraint that shared state be `no pending request` during its scheduled interval. There are no shared resources although communication re-

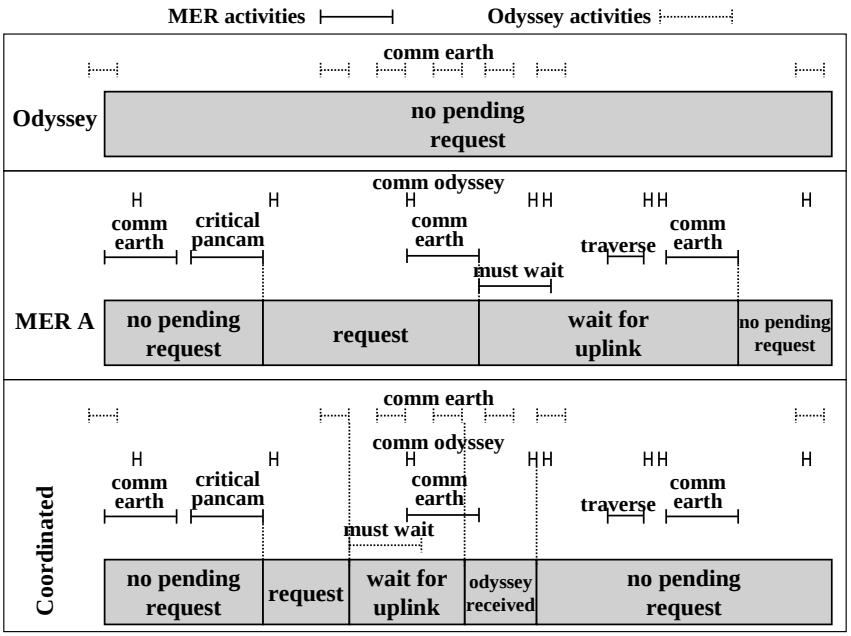

Figure 7: Downlink/uplink shared state for MER A. From top to bottom, Odyssey's initial view, MER A's initial view, and the common view after coordination.

quests from a MER have effects on many local resources of both the MER and the orbiter. All of the shared activities have active master and passive slave roles. The MERs and obiters both take the master role for activities labeled for them in Figure 6.

CASPER planners for each of the MERs and orbiters first build their three-day plans separately to optimize the timely delivery of priority weighted science data, resolving any local constraints on memory, power, battery energy, *etc*. The three-day schedules constitute over 900 tasks for each MER and over 1300 for each of the orbiters with 30 state/resource variables for each MER and 22 for the orbiters. Planning is slowed by a factor of 440 to account for differences between a desktop workstation and a radiation-hardened flight processor. Communication for coordination is restricted to times when the orbiters pass overhead. With the exception of Mars Express, the orbiters pass overhead once every eight hours. Because of its irregular orbit, Mars Express sees the rovers only once every 96 hours. Because of this, we actually used no consensus window for communication with Mars Express, thus, forcing the planners to resolve conflicts during image transmission.

When coordination begins, the planners send their communication requests to the other planners while optimizing their plans. Before these updates are received, the initial views of the shared uplink status are shown in Figure 7. The MERs begin with conflicts with their traverse tasks because the uplink has not yet been received from Earth. The coordination algorithm commands the planners to repetitively process shared task updates, replan to resolve conflicts by recomputing the shared state and modifying scientific measurement operations to adjust for the increased power and memory needs, and send task updates. After a minute and a half, MER A, B, and Odyssey agree on routing the downlink and uplink through Odyssey to get the uplinked commands in time for the traversal on different days. he resulting shared state is shown at the bottom of Figure 7. The planners reach consensus that coordination is complete and sleep while waiting for task updates.

Among other failed communication attempts, we triggered an anomaly in MER A's plan causing it to cancel its first day's tasks and shift the entire schedule forward a day. Before sending the updated shared tasks, replanning was issued to resolve local constraints to avoid propagating inconsistent state information to Odyssey. All conflicts were resolved in a few seconds except the traverse conflicts with a `wait` state. Then MER A sends a task update to restart coordination. Coordination completes in less than a minute with data again being routed through Odyssey.

While we have only experimented with simple protocols, this application of SHAC to the Mars scenario shows how planners can coordinate during execution while making minimal concessions to ideal plans and responding to unexpected events. In the next section, we discuss how SHAC builds on related work and discuss new research challenges for decentralized, coordinated planning.

7 Discussion and Related Work

Conflicts among a group of agents can be avoided by reducing or eliminating interactions by localizing plan effects to particular agents [Lansky, 1990], and by merging the individual plans of agents by introducing synchronization actions [Georgeff, 1983]. In fact, planning and merging can be interleaved [Ephrati and Rosenschein, 1994]. Earlier work studied interleaved planning and merging and decomposition in

a distributed version of the NOAH planner [Corkill, 1979] that focused on distributed problem solving. More recent research builds on these techniques by formalizing and reasoning about the plans of multiple agents at multiple levels of abstraction to localize interactions and prune unfruitful spaces during the search for coordinated global plans [Clement and Durfee, 2000].

DSIPE [desJardins and Wolverton, 1999] employs a centralized plan merging strategy for distributed planners for collaborative problem solving using human decision support. Like our approach, local and global views of planning problem help the planners coordinate the elaboration and repair of their plans. DSIPE provides insight into human involvement in the planning process as well as automatic information filtering for isolating necessary information to share. While our approach relies on the domain modeler to specify up front what information will be shared, SHAC supports a fully decentralized framework and focuses on interleaved coordination and execution.

In many ways this work is following the Generalized Partial Global Planning approach to using a mix of coordination protocols tailored for the domain [Decker, 1995]. SHAC offers an alternative framework for separating implementation of these mechanisms from the planning algorithms employed by specific agents. Unlike GPGP, SHAC provides a modular framework for combining lower-level mechanisms to create higher-level roles and protocols. Our future work will build on GPGP's evaluations of mechanism variations to better understand how agents should coordinate for domains varying in agent interaction, communication constraints, and computation limitations.

Finally, TEAMCORE provides a robust framework for developing and executing team plans [Tambe, 1997; Pynadath *et al.*, 1999]. This work also offers a decision-theoretic approach to reducing communication within a collaborative framework. Research is needed to investigate the integration of coordinated planning with robust coordinated execution.

An assumption commonly made in multiagent research is that agents will be able to communicate at all times reliably. In the Mars scenario, the spacecraft communicate with each other in varying time windows and frequencies, and the two MERs can never directly talk to each other. Establishing consensus on beliefs and intentions is impossible without certain communication guarantees [Mullender, 1995]. Understanding the communication properties that make consensus possible and the overhead for establishing consensus is critical for multiagent research.

8 Conclusion

We informally described a continual coordination problem and its challenges. SHAC addresses the problem as a general planner-independent continual coordination algorithm and a framework for designing and evaluating role-based coordination mechanisms. We described its capabilities and gave examples of higher-level mechanisms built on these capabilities. In addition, we have presented an algorithm for determining the time to reach consensus given a protocol and communications constraints. While our future work is aimed at evaluating the benefits of different protocols for different classes of multiagent domains, we validate our approach in coordinating five simulated spacecraft experiencing unexpected events.

References

[Chien *et al.*, 2000] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proc. ECP*, pages 300–307, 2000.

[Clement and Durfee, 2000] B. Clement and E. Durfee. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proc. ATAL*, pages 213–227, 2000.

[Corkill, 1979] D. Corkill. Hierarchical planning in a distributed environment. In *Proc. IJCAI*, pages 168–175, 1979.

[Decker, 1995] K. Decker. *Environment centered analysis and design of coordination mechanisms*. PhD thesis, University of Massachusetts, 1995.

[desJardins and Wolverton, 1999] M. desJardins and M. Wolverton. Coordinating a distributed planning system. *AI Magazine*, 20(4):45–53, 1999.

[Ephrati and Rosenschein, 1994] E. Ephrati and J. Rosenschein. Divide and conquer in multi-agent planning. In *Proc. AAAI*, pages 375–380, July 1994.

[Georgeff, 1983] M. P. Georgeff. Communication and interaction in multiagent planning. In *Proc. AAAI*, pages 125–129, 1983.

[Grosz and Kraus, 1996] B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–358, 1996.

[Kraus *et al.*, 1998] S. Kraus, K. Sycara, and A. Evanchik. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104:1–70, 1998.

[Lansky, 1990] A. Lansky. Localized search for controlling automated reasoning. In *Proc. DARPA Workshop on Innov. Approaches to Planning, Scheduling and Control*, pages 115–125, November 1990.

[Mullender, 1995] S. Mullender. *Distributed Systems*. Addison-Wesley New York, 1995.

[Pynadath *et al.*, 1999] D. Pynadath, M. Tambe, N. Cauvat, and L. Cavedon. Toward team-oriented programming. In *Proc. ATAL*, 1999.

[Tambe and Jung, 1999] M. Tambe and H. Jung. The benefits of arguing in a team. *AI Magazine*, 20(4), 1999.

[Tambe, 1997] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[Yokoo and Hirayama, 1998] M. Yokoo and K. Hirayama. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. on KDE*, 10(5):673–685, 1998.