Continual Coordination of Shared Activities

Bradley J. Clement and Anthony C. Barrett
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
{bclement, barrett}@aig.jpl.nasa.gov

Abstract

Interacting agents that interleave planning and execution must reach consensus on their commitments to each other. For domains with varying degrees of interaction and different constraints on communication and computation, agents will require different coordination protocols in order to efficiently achieve their goals. ShAC (Shared Activity Coordination) is a framework for designing coordination protocols and an algorithm for continually coordinating agents using these protocols during execution. We show how a variety of protocols can be constructed using this framework and describe how ShAC coordinates two rovers and an orbiter in a simulated Mars scenario.

Introduction

Agents based on behavioral systems or social laws (Shoham & Tennenholtz 1992) can make short-sighted coordination decisions, leading to irreversible effects that prevent agents from reaching their longer-term goals. Planning is commonly used to project and resolve conflicts with future activities. When interleaving planning and execution, an agent must be able to adjust its planned activities as it gathers information about the environment and encounters unexpected events. Interacting agents must coordinate these adjustments in the context of commitments with each other. The work presented here addresses how these agents can interleave coordination and execution. The ultimate goal of this research is to enable interacting agents to autonomously adjust their coordination protocols with respect to unexpected events and changes in constraints on communication and computation so that the agents can most efficiently achieve their goals. This paper presents a framework for designing coordination protocols and an algorithm for continually coordinating agents using these protocols during execution.

Our approach, called Shared Activity Coordination (ShAC), provides a general algorithm for interleaving planning and the exchange of plan information as shared activities. Agents coordinate their plans by establishing consensus on the parameters of shared activities. Figure 1 displays this consensus on parameters as *equals* constraints between agents' local views of the shared activity. The vertical box over each planner's schedule represents a *commit window* that moves along with the current time. Activities in this window must be passed on to the execution system, which sends state updates to the planner. Consensus must be established for shared activities before this window to avoid violated commitments between agents. Consensus is hard to establish if all agents sharing an activity modify its parameters at the same time. Thus, agents participate in different coordination roles that specify which agent has control of the activity.

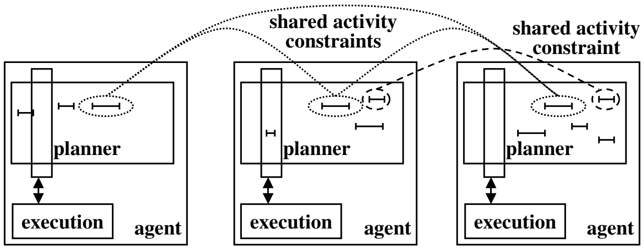

Figure 1: Activities shared among continual planners

ShAC's ability to continually coordinate depends on interleaved planning and execution. As a result, the planner must be able to respond to execution failures and state updates from the execution system. Our implementation interfaces with one such continual planning system, CASPER (Continuous Activity Scheduling Planning Execution and Replanning) (Chien *et al.* 2000a). 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 the execution system.

First we describe the shared activity model, the ShAC algorithm, and its interface to the planner. Then we specify some generic roles and protocols using the ShAC framework that build on prior coordination mechanisms. These mechanisms include a mapping of distributed constraint satisfaction and argumentation approaches to distributed planning. Then we describe how our current implementation of ShAC is used to coordinate the communication of two rovers and an orbiter in a simulated Mars scenario. We follow with future research needs revealed in this scenario and comparisons to related work.

ShAC

ShAC is implemented as a module that commands the planner of each agent and handles communication with other agents. ShAC keeps track of shared activities and constraints on these activities.

Shared Activities

Consider an example of a shared communication activity. One agent serves the role of a sender and another the role of a receiver. 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. Planning decisions affect shared activities by altering the values of parameters. For example, a planner can reschedule an activity by changing its $start\_time$ parameter.

A shared activity is a tuple $\langle identifier, parameters, agent\ roles, protocols, decomposition \rangle$. The parameters are the shared variables and current values over which the agents must reach consensus. Our implementation defines boolean, integer, floating point, and string types. Agent roles are the local activities assigned to each agent that correspond to the shared activity. These roles can have different conditions and effects as specified by the local planning model. The shared parameters map to local parameters in the role activity. Protocols are the mechanisms assigned to each agent (or role) that allow the agents to

- change constraints on the shared activity,
- change the set of agents assigned to the activity, and
- change roles and protocols for each agent.

Multiple protocols can be defined for a shared activity to combine their capabilities. Constraints will be described in the next section, and a variety of protocols will be defined in the Protocols section.

The decomposition is the set of subactivities in the decomposition of the agent's local role activity that all agents share. This is needed to establish consensus on shared subactivity choices that may be selected or eliminated from the set of subactivities.

Constraints

Constraints are created by agents' protocols to restrict sets of values for parameters (*parameter constraint*) and permissions for manipulating the parameters, changing constraints on the parameters, and scheduling shared activities (*permission constraint*). 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. By only permitting one agent to have planning capabilities at a time, a protocol can avoid thrashing among agents that try to resolve shared activity conflicts in different ways.

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 & 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. As mentioned before, scheduling constraints can be represented as constraints on a start time integer parameter.[1]

Permissions 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
  - duration - change duration of task
  - delete - remove from plan
  - choose decomposition - select shared subactivity of an *or* activity
- add - add to plan[2]
- constrain - send constraints to other agents

Parameter permissions can be specified for each parameter separately. The permissions categorized under the parameters permission are potentially special parameters that all activities have. A protocol can change permissions of shared activities for any agent during coordination.

Coordination Algorithm

Figure gives a general specification of the ShAC algorithm. ShAC is implemented separate from the planner, so 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.

Note that on each pass of this loop, there may be no conflicts to warrant altering the plan, no state updates to revise the projection, no activities to release, or no messages or other ShAC information to act upon. So, some steps could execute several times before others execute once. This algorithm could be interpreted to run all steps in parallel with each other. Each separate parallel step could be looping at a different rate, either naturally or as configured. The best way to configure this will depend on the domain and the agents' sensing and computation capabilities. Although a domain designer can make guesses and experiment through simulation, future work is needed to understand how this configuration should be done in general.

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 is an influence of interfacing ShAC to the CASPER planning system, which grounds the timing of all activities (Chien *et al.* 2000a). ShAC would need a new representation to handle partial order constraints on activities.

[2] 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).

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 and $constraints$;
   - else
     – apply simple consensus protocol.
5. Communicate changes in $shared\_activities$ and $constraints$.
6. Update $shared\_activities$ and $constraints$ based on received communications.
7. Go to 1.

Figure 2: Shared activity coordination algorithm

This interaction with the execution system becomes more complicated when agents share tasks. ShAC must make sure that any shared activity is released by all agents with consensus on the start time and other parameters of the task. Ideally the agents should establish consensus before the commit window. If they cannot, then there should be a simple protocol that allows them to quickly establish consensus (e.g. use the task information of the highest priority agent that shares the task and has modification permissions), and the task must be modified within 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 consensus window extends forward. When a shared activity moves into the window, the agents switch to the simple consensus protocol to try and reach consensus before the activity moves into the commit window.

ShAC requires several capabilities from the planning system to which it interfaces. If the system is not a continual planner, and coordination is to be done in batch before execution, then steps 1 and 3 are not necessary in the algorithm description. In this case, ShAC requires the following capabilities of the planner:

- grounds start times and durations of activities[3],
- add new activities received from other planners,
- update parameters of activities changed by other planners,
- enforce constraints during planning, and
- generate parameter constraints for activities it fails to schedule.

[3]This is because ShAC has not yet been extended to handle flexible or partial order time constraints.

For continual coordination during execution, the planner must additionally be able to

- integrate state updates from the execution system,
- project changes in the current plan, and
- issue near-term activities to the execution system.

Probably the most stringent requirement is that the planner be able to alter its plan based on state updates and changes made by other agents to shared activities. Being able to handle unexpected events, however, is a necessary property for interleaving planning and execution. Iterative repair planners focus on this capability and make good candidates for this coordination architecture. CASPER is one such planner and provides all of the capabilities we list above (Chien *et al.* 2000a).

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 (of the previous section), the protocol determines what to communicate and how to process communication as described in the algorithm and the Shared Activities section. A protocol must specify 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/protocols of sharing agents

We will define protocols according to these four methods. Even if a protocol does nothing for these methods, the ShAC algorithm still has several capabilities. An initial assignment of agents and their permissions to shared activities can provide the following capabilities. A domain modeler uses a language for defining shared activities to specify these initial assignments to provide a mix of capabilities resulting in basic protocols upon which others can be built.[4]

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.

resource sharing Sharing agents can have identical conditions and effects on shared states or resources.

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

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

[4]The modeling language is outside the scope of this paper but is inspired by team-oriented programming for STEAM (Pynadath *et al.* 1999; Tambe 1997).

So, a default protocol implements does nothing for each of the protocol methods but retains the above listed capabilities. In our implementation this default protocol is a base class from which subclasses build on these capabilities by defining one or more of the protocol methods. The following sections describe some of these protocol subclasses.

Argumentation

Argumentation is a technique for negotiating joint beliefs or intentions (Kraus, Sycara, & Evanchik 1998). Commonly, one agent makes a proposal to others with justifications. The others evaluate the argument and either accept it or counterpropose with added justifications. This technique has been applied to teamwork negotiation research to form teams, reorganize teams, and resolve conflicts over members' beliefs (Tambe & Jung 1999). This technique can be used in coordinated planning to establish consensus on shared activities.

A shared activity and associated parameter values are the proposal or counterproposal. Justifications are given as parameter constraints (not necessarily on the same parameters proposed). A proposal is a change to a shared activity that does not violate any parameter constraints. A counterproposal is a change that does violate another agent's parameter constraints. Protocol method 2 must be implemented to provide the parameter constraint justifications for proposals and counterproposals. This involves noting changes in the shared activity and (if changed) generating constraints describing the parameter values consistent for the agent. The parameters for which constraints are generated is domain dependent since the argument for a proposal is domain dependent. The protocol could be implemented as follows.

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

As an 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 accomodate the proposal and counterpropose with their own interval restrictions if replanning cannot accomodate 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 counterpropose.

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's protocol only needs to implement protocol method 3 to choose the subordinates that will perform the task. So, when delegating a new activity or redelegating an existing activity, method 3 clears the list of agent roles and assigns the delegated agent the subordinate role.

Criteria for determining when to delegate to a subordinate and to which subordinate to delegate may vary among domains. A simple method is to delegate whenever the shared activity has not been assigned to an agent (no agent roles exist). Redelegation occurs when a subordinate fails to accomodate the shared activity in its plan and rejects it by removing itself from the activity's agent roles (protocol method 3 for the subordinate role). If the delegator has information about the agents' plans (e.g. from other shared activities or communicated constraints), it may be able to judge which agent is best to delegate the activity according to its capability or estimated performance. This protocol can be simply varied by assigning delegators and subordinates different master or slave roles, or having multiple subordinates with subroles in the shared 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 $agent\ roles$

Asynchronous Weak Commitment

Multi-asynchronous weak commitment is a distributed constraint satisfaction algorithm that enables agents, each with a set of variables, to satisfy constraints between variables across and within agents (Yokoo & Hirayama 1998). Agents are prioritized, and their variables each initially have a zero priority. The values of lower priority variables are modified to satisfy constraints with values chosen for higher priority variables (with agent priorities as a tie breaker). If there is no value that satisfies the constraints, then the governing agent selects a value that minimizes violations with lower priority variables and raises the priority of the variable to one higher than the highest priority of the variables with which it has constraints, making the variable the highest ranking with its neighbors. The failing agent also sends a *no-good* to its neighbors, communicating the values of the subset of variables making the variable unassignable.

This protocol can be adapted for planning agents. The variables are shared activity parameters. The constraints are $equals$ relations among agents sharing the activities. An agent's protocol must keep track of a priority of the shared activity and those of the other sharing agents (as well as the agents' priorities). These priorities can be parameters of the shared activity. A no-good message is a set of parameter constraints. Below are the protocol methods for updating permission constraints and rank and generating no-good parameter constraints.

Asynchronous Weak Commitment method 1

- if have highest priority
  - remove self's modification permissions

- else
  - give self modification permissions (e.g. move and delete)

Asynchronous Weak Commitment method 2

- if cannot resolve conflicts/threats locally and with respect to constraints of higher ranking agents
  - set rank parameter of self to highest rank of sharing agents plus one
  - generate parameter constraints (no-good) describing locally consistent values

So, the protocol only needs to handle the agent's own permission and parameter constraints. When an agent has permissions to modify the shared activity, the planner is responsible for scheduling (choosing parameter values of) the activity that satisfy the parameter constraints higher ranking agents while trying to respect those of lower-ranking agents. We do not mention this aspect of the planner interface as part of ShAC, but our implementation subclasses planner interfaces to correspond to protocols. The implentation has special purpose heuristics for guiding CASPER's planning to handle these kinds of constraints. Heuristic capabilities are discussed in detail in the ASPEN planner upon which CASPER is built (Chien *et al.* 2000b).

The asynchronous weak commitment algorithm for DCSPs is shown to be sound and complete–the agents are guaranteed to converge to valid assignments of values to variables if they exist. In ShAC there is no guarantee of completeness (convergence). This is becasue we do not restrict the planners to be complete. Since continual planning requires reactivity to state changes and failures, completeness is difficult to ensure in real-time. Future work is needed to determine how AWC can be combined with complete planners to ensure convergence.

Constraint-Based Conflict Resolution

For this protocol, the agents initially have no permissions to modify a proposed shared activity. They broadcast any parameter constraints to the sharing agents as the planner schedules other local or shared activities around the shared activity while trying to meet others' constraints as possible. After some time period, or once the agents have converged on a set of constraints (not guaranteed), the agents switch to another protocol (e.g. argumentation) potentially reinstating permissions and negotiate final parameter values or delete the activity. Detecting convergence on parameter constraints requires the agents to reach consensus on the current set of constraints. Establishing consensus is described in (Mullender 1995). We discuss the role of consensus in coordination in the section on Discussion and Related Work. The protocol must implement method 2 for generating parameter constraints and method 4 to switch protocols.

Constraint-Based Conflict Resolution method 2

- if cannot resolve conflicts/threats involving shared activity
  - update parameter constraints describing locally consistent values

Constraint-Based Conflict Resolution method 4

- if reached consensus on constraints or $time\_elapsed >$ threshold
  - switch to protocol for resolving conflicts

Round-Robin

A round-robin approach to establishing consensus on a shared activity involves rotating a master role by changing permission constraints. Only one agent may modify the activity at a time and once finished, the agent can turn off its own permissions and turn them on for another agent (while sending out the update). Only protocol method 1 needs to be implemented to switch permissions—role switching is not necessary.

Round-Robin method 1

- if have modification permissions
  - update $time\_elapsed$
  - if finished planning or $time\_elapsed >$ threshold
    - remove self's modification permissions (e.g. move and delete)
    - add modification permissions for next agent
    - set $time\_elapsed$ to 0

In this method, the agent may determine that it is "finished planning" if there are no remaining local conflicts/threats or if the plan is sufficiently optimized. $time\_elapsed$ can be a parameter of the shared activity that is updated by the current master agent. A simple adaptation of this protocol could have agents generate parameter constraints in the innermost `if` statement. These constraints would need to be updated as the planner reschedules other activities based on changes to the shared activity by subsequent agents.

Centralized Conflict Delegator

Here, a single agent serves in a passive delegator role for a set of shared activities. The delegator models all shared resources and, thus, keeps track of all conflicts for a group of active subordinates. Subordinates do not share activities with each other. The delegator assigns conflicts to different agents by delegating tasks involved in conflicts to different subordinates and also sending the subordinates the corresponding parameter constraints it generates indirectly from the activities it shares with other subordinates. This protocol can subclass from the basic delegation protocol. The difference is in how it chooses the agent to whom to delegate the activity. Below we define this procedure, which is called from Delegator method 3 in the Delegator section. This function ensures that agents are not modifying the same activities or working on the same conflicts (in order to avoid race conditions).

ChooseSubordinate method

- sort agents in increasing order of times this activity was delegated to them
- for each agent
  - if not delegated any activities involved in conflicts with this one
    - return agent
- return first agent

Application to Mars Scenario

Now we describe how ShAC is applied to a simulated scenario involving two Mars Exploration Rovers (MERs) and a Mars Odyssey orbiter. 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. We will apply some of the previously defined, more sophisticated protocols to this domain in our future work.

The MERs (MER A and MER B) and Odyssey can communicate with Earth directly, but the MERs can optionally route data through Odyssey, which communicates 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 Odyssey to determine how to most quickly get a response from ground after sending an image of the surrounding area.

Each MER has a communication state shared with Odyssey 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 3. 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 a `wait for uplink` state 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 requests from a MER have effects on many local resources of both the MER and Odyssey. All of the shared activities have active master and passive slave roles. MER and Odyssey both take the master role for activities labeled for them in Figure 3.

CASPER planners for each of the MERs and Odyssey first build their three-day plans separately to optimize science data return, resolving any local constraints on memory, power, battery energy, *etc*. The three-day schedules constitute over 600 tasks for each MER and over 1400 for Odyssey with 30 state/resource variables for each MER and 22 for Odyssey.

When coordination begins, the planners send their communication requests to the other planners. Before these updates are received, the initial views of the shared uplink status are shown in Figure 4. 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

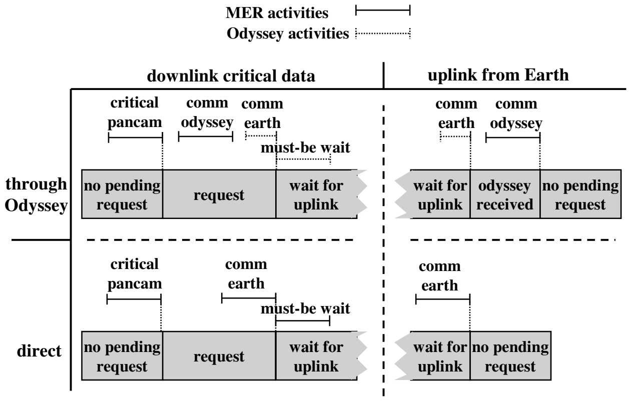

Figure 3: Downlink/uplink states for a rover

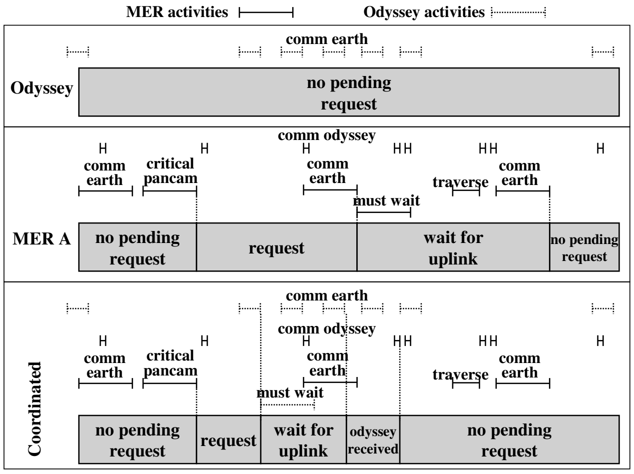

Figure 4: Downlink/uplink shared state for MER A. From top to bottom, Odyssey's initial view, MER A's initial view, and the common view after coordination.

agree on routing the downlink and uplink through Odyssey to get the uplinked commands in time for the traversal on different days.[5] The resulting shared state is shown at the bottom of Figure 4. The ASPENs reach consensus that coordination is complete and sleep while waiting for task updates.

Then 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 min-

[5]Odyssey's planner ran on a SunBlade 1000, and the MERs ran on a Sparc Ultra 60 and 80.

imal 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.

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 & 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 & Durfee 2000). While this is a centralized approach, work is needed to apply these techniques that leverage abstraction in a decentralized framework to reduce communication and computation during coordination. Abstract plan information can even automate the discovery of agent relationships that our approach pushes off on the domain modeler.

DSIPE (desJardins & 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 classes of domains varying in agent interaction, communication constraints, and computation limitations.

Grosz's shared plans model of collaboration presents a theory for modeling multiagent belief and intention (Grosz & Kraus 1996). ShAC's model and manipulation of shared activities provides basic mechanisms for agents to communicate and establish beliefs, intentions, and goals for itself or a group. Using ShAC to reason about the mental states of agents, the shared plans model and work based on BDI (belief-intention-desire) models of agents (Rao & Georgeff 1995) can be exploited in ShAC.

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 patterns that make consensus possible and the overhead for establishing consensus is critical for multiagent research.

Conclusion

We have introduced shared activity coordination as an approach to designing role-based coordination mechanisms for planning agents. ShAC provides several coordination capabilities upon which we have specified several higher-level coordination protocols, including a mapping of a distributed constraint satisfaction algorithm into distributed planning. We have also described an algorithm for continually coordinating planning agents during execution using these protocols. While our future work is aimed at evaluating the benefits of these protocols for different classes of multiagent domains, we validate our approach in coordinating three simulated spacecraft in the presence of an unexpected event.

Acknowledgments

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.

References

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

Chien, S.; Rabideu, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000b. Automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps*.

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

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

Decker, K. 1995. *Environment centered analysis and design of coordination mechanisms*. Ph.D. Dissertation, University of Massachusetts.

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

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

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

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

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

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

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

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

Rao, A. S., and Georgeff, M. P. 1995. BDI-agents: From theory to practice. In *Proc. ICMAS*.

Shoham, Y., and Tennenholtz, M. 1992. On the synthesis of useful social laws for artificial societies. In *Proc. AAAI*, 276–281.

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

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

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