CLEaR: A Framework for Balancing Deliberative and Reactive Control

Forest Fisher, Daniel M. Gaines, Tara Estlin, Steve Schaffer, Caroline Chouinard

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
{firstname.lastname}@jpl.nasa.gov

Abstract

A major challenge in developing robotic applications for real-word problems is that many domains include tight resource and temporal constraints coupled with uncertainty in how much resource and time will be required to perform a task. We have developed the CLEaR framework to address this challenge. CLEaR unifies planning and execution processes to increase the responsiveness of a robotic agent operating in these types of environments. This unified approach is realized by extending the traditional three-tier robotic control architecture with an Execution-Time Plan Manager, an Atomic Resource Manager (ARM) and an Execution-Time Query (ETQ) capability. Through the interaction of these components, CLEaR is able to (1) reduce the need to replan, (2) detect the need to replan earlier, and (3) replan before entering a failed state.

Introduction

Robotic agents performing under hard resource and time constraints in uncertain environments require careful balancing of both deliberative and reactive reasoning [Knight, et al, 2001]. As in most domains with uncertainty, a task may fail or produce unexpected results leading to plan failures. If the robot is also under hard time deadlines and resource constraints, a task requiring a different time or resource allocation than planned could cause failure at future points in the plan. In some cases, the robot may be able to retry a failed task, use more time or take up more of a resource without causing a problem.

Consider for example, a Mars exploration rover that must pick up a rock. If it fails on its first attempt, it may want to try again. However, doing so could lead to other problems at later stages of the plan. If the rover spends too much time trying to complete this task, it may miss another deadline, such as taking an image while the sun is in a particular position in the sky. Or, it may use up too much of some resource, such as energy, resulting in the inability to perform other critical tasks. The challenge is to determine whether or not a change in time or resource usage will cause a problem so that the rover can take appropriate action, and to identify and fix the conflicts in the plan without preventing the rover from meeting other deadlines. In this example, a deliberator is used to project current resource and time usage into the future, detect problems and make repairs. An executor uses more reactive reasoning to deal with unexpected events and perform low-level control. The rover needs both of these capabilities to successfully operate in this environment.

Most of the robotic applications in the literature have not been confined by hard resource constraints and strict time deadlines; consequently little work has been done in this area. However, there has been a growing awareness of these issues in recent years. At NASA, almost all the robotic space exploration domains involve uncertain environments with deadlines and tight resource constraints.

In pursuit of developing high-level control software capable of addressing these issues, we have developed the CLEaR (Closed Loop Execution and Recovery) control software/framework. CLEaR provides a unified framework for performing planning, scheduling and execution by balancing both deliberative and reactive reasoning. In most related approaches to robotic control, the planning and execution components are treated as black box functions that do not interact in real-time. Our approach differs in that both the planning and execution functionalities share the responsibility for decision-making and resource management.

In our system the unified planning and execution responsibilities are realized through three means of increased interaction and information sharing between the deliberative and reactive functions:

1. The executor provides soft-real-time state, resource and time updates enabling the deliberator to anticipate problems and replan if necessary.
2. The deliberator provides rapid response to queries about time and resource usage variations, thus enabling the executor to manage a task that is behaving unexpectedly.
3. The executor uses execution time resource knowledge while managing tasks.

By enabling the long-term deliberation and the short-term reactive execution functionalities to share information on a more frequent basis, the system can: (1) reduce the need to replan, (2) detect the need to replan earlier and (3) replan while continuing to execute valid portions of the

plan without entering a failed state. In other words, the system can circumvent as many failure situations as possible without impacting plan execution. By achieving these capabilities we are able to produce a robotic agent control system capable of goal-based commanding in an uncertain environment while adhering to hard resource and time constraints.

Our framework for balancing deliberation and reaction has been motivated by several NASA space exploration domains. The most significant influence has been Mars surface exploration with autonomous rovers, especially the proposed Mars 2009 Smart Lander mission. In the next section we will describe this mission and illustrate how the mission provides challenging time and resource constraints for an autonomous robot. We will describe how we have designed CLEaR to deal with these types of challenges and then present a case study illustrating how CLEaR will enable a rover to successfully deal with these challenges.

Mars (2009) Smart Lander Rover scenario

In 1997, JPL successfully completed the first mission to explore Mars' surface with a mobile robotic platform (Sojourner rover). During the mission, human ground teams performed nearly all deliberative decision-making including the determination of resource bounds. While the mission was a landmark in space exploration and provided valuable science data, it required intensive human interaction and explored a very small region of terrain.

In 2009, JPL plans to send another mobile robotic platform to Mars to perform numerous geological surface experiments. This mission is currently called the Mars Smart Lander mission and represents a significant increase in scale with respect to mission duration, science return and terrain covered. Figure 1 provides an overview of the mission. The mission objectives are to explore the landing site and make long-range traverses to two additional geological science locations where the robot will perform more science data gathering. The rover will have limited resources, such as power and RAM, to complete these goals. It will also be under tight time constraints in order to complete the ambitious objectives and meet mission requirements, such as ground communication windows.

There will be communication with Earth at the beginning and the end of each Martian day. In the morning session, the goals for the day are uploaded to the rover and additional data will be down-linked. In the evening, the day's data is down-linked. This data includes panoramic images used in selecting future goals.

This scenario has two modes of operation.. The first being the geological science location operations, and the second being the long-range traverses between those locations. During the first mode, the role of high-level autonomy software will primarily involve resource management (mainly power, memory and time) and robust execution.

During the second mode, the rover is expected to make long-range traverses averaging 600m/day. This distance is

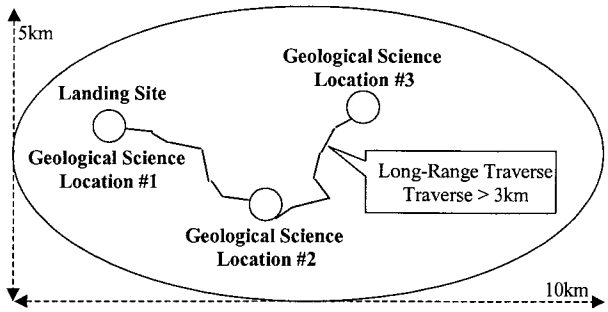

Figure 1: Mars Smart Lander Scenario

well beyond the "line of sight" of the ground operations team based on images down-linked from the previous day. Therefore the traverse will require significant onboard autonomy. Further motivating the need for high-level goal-based autonomy is that the rover should perform as much opportunistic traverse science as possible without impacting the progress of the 3km long-range traverses.

Unified Planning and Execution Framework

Current practice for rover operations, as used on the Pathfinder mission [Mishkin, et al, 1998] and planned for the upcoming 2003 Mars Exploration Rover (MER) mission, is to perform nearly all decision making remotely from earth. When the rover encounters a situation that deviates from its uploaded command sequence, the fault protection software will attempt to resolve the problem. Failing that, the rover enters *safe-mode* and must: wait for a communications opportunity, transmit the state of the rover and imagery of the environment back to Earth, and wait for a new command sequence. Depending on when the next communication window is scheduled, this can waste considerable time. Further, these rovers are solar powered and can only perform major functions for a few hours per day (typically 4-6 hours). Placing the rover in *safe-mode* can easily cause the loss of a full day of operations. Because the mission cannot be extended, falling behind schedule due to execution failures results in reduced science return.

While this style of operations reduces development cost and simplifies testing of flight control software, it adds to the complexity of mission operations. This increases cost and severely limits the rover's in-situ capabilities.

From an automation standpoint part of what limits rover operations performance is that the decision-making process has traditionally been separated from the execution process. To address this several systems have colocated the deliberative-planning and execution capabilities, to dramatically increase the rover's responsiveness and reduce the need for the rover to be put into *safe-mode*.

Most of these systems can be classified as three-tiered control architectures [Gat 1998]. Under a three-tiered system the deliberative planning and reactive execution components are colocated but tend to function independently typically in a black-box integration. These

architectures get their name from a stack-like partitioning of the system into three functional components. The top tier provides deliberative function, the middle tier performs reactive execution, and the bottom interfaces to the hardware controllers. Generally, the higher up in the stack, the greater the level of abstraction at which the components functions and the longer it takes to perform. The top tier is usually reserved for search algorithms. In the event of an execution failure, this approach can reduce the time the rover waits for ground intervention by facilitating replanning onboard.

One drawback of traditional three-tiered approaches is that they do not instigate planning prior to an execution failure. In order to replan and thus preempt execution failures, it is necessary to provide the deliberator with frequent state, resource and temporal updates. These can then be propagated through the plan to predict future inconsistencies. If the deliberator is able to incrementally resolve these conflicts while executing valid portions of the plan, then the robotic agent will be more responsive to unexpected events [Zweben, et al 1994; Minton, Johnston 1998]. We refer to this capability as continuous planning.

In our implementation of this framework, we use CASPER (Continuous Activity Scheduling Planning Execution and Replanning) as the continuous planner [Chien, et al, 2000a, 2000b]. CASPER provides the Deliberator and Execution-Time Plan Manager components depicted in Figure 2. The Executor component is provided by TDL (Task Description Language), a robust task level execution framework [Simmons, Apfelbaum 1998].

The CLEaR framework is distinct from other three-tier architectures because it provides increased interaction and information sharing between the executor and the deliberator [Gat 1998, 1992; Bonasso, et al, 1997]. This is partly realized by the use of a continuous planner combined with frequent updates from the executor to the deliberator

Two other areas of increased interaction and information sharing are provided by the executor's ability to: (1) make decisions on how to execute a task by querying the deliberator to determine if a given execution will cause a plan failure and (2) consider execution time resource knowledge in deciding on task expansions. These last two capabilities are provided by the Execution Time Query (ETQ) manager and the Atomic Resource Manager (ARM), also depicted in Figure 2. In the following two sub-sections, we describe these components.

ARM: Atomic Resource Management

Motivations and Design Goals

There are certain types of activities that require a resource intermittently during their execution. For example, while a rover is navigating, it will occasionally take images to detect and avoid obstacles in its path. Although navigation requires the camera, it does not use it continually. In fact, after each image, it can make a rough estimate of when it needs the camera again. This can be done because, given an image, the navigation activity

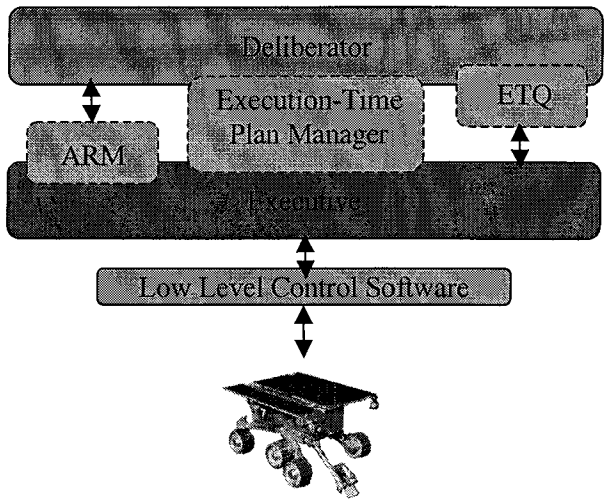

Figure 2: CLEaR Framework Diagram

determines how far it can safely travel before taking another image. As a result, the camera will become available at different times throughout the navigation, and it would be nice if other activities could take advantage of this.

In fact, within the context of the Mars 09 Smart Lander mission, there is a need for such a capability to enable opportunistic traverse science during a long-range traverse. Traverse science uses the camera to take images at different times during the traverse to look for items of interest. Like navigation, traverse science does not need the camera continually throughout the traverse and could use the camera when not in use by navigation.

In general, we may have several activities that each make intermittent use of a particular resource. If we knew ahead of time when each requires the resource and for how long we could use deliberative scheduling techniques to create a plan to avoid resource conflicts while executing these activities. Unfortunately, for some activities, such as navigation, we cannot accurately predict when the resource will be needed. Furthermore, the accuracy of our prediction will decrease as we attempt to predict uses further in the future. As a result, an activity may use the resource earlier or later than expected, and once the activity has the resource, it may require it for a duration different from what it had originally anticipated.

Given these conditions, scheduling such activities is challenging. Previous approaches for dealing with these issues include the following. First, a planner could avoid the problem by refusing to schedule activities concurrently if they require the same resource, regardless of whether or not they only require the resource intermittently. The downside to this is that you are limiting the robot's capabilities, and in some applications concurrent activities are required to complete a goal. A second approach would be to form a deliberative schedule for these activities based on rough estimates on the frequency and duration that each activity will use the resource. The disadvantage here is that it is very unlikely that during execution the activities will

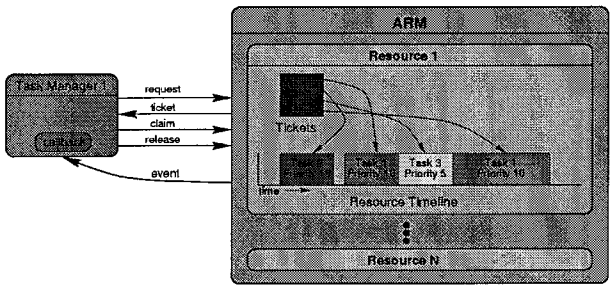

Figure 3: Design of ARM

use the resources as predicted. This could be handled by performing rescheduling within the planner as it gets new updates on actual resource usage or by allowing the executive to preempt lower priority tasks whenever there is contention for the resource. The former approach is likely to result in thrashing within the schedule as information changes. Both approaches are likely to lead to a large number of preempted activities. A third approach is to create special executive task managers for each combination of activities that may need to run in parallel. Each such manager would be designed to arbitrate resource usage among these particular activities.

Instead, we have chosen to deal with these challenges by developing a resource manager for use within the executive. Because the executive needs to be responsive to unexpected changes, our primary design goal is to keep the resource manager fast so that it can quickly respond to requests. Therefore, we will favor simpler designs and algorithms to reduce computational complexity.

Because the predictions on when and how long a resource will be used is uncertain, the resource manager must be able to quickly react so that high priority activities can have access to the resource when it is needed. However, the resource manager should make use of predicted information when available to try and reduce the number of times it must preempt another activity. Thus, our secondary goal is to balance the use of deliberation and reaction, where deliberation takes advantage of predicted information and reaction to deal with unexpected changes in resource usage.

Design of ARM

For our first implementation of ARM, we decided to address only atomic resources. An atomic resource can be used by at most one task at a time and is either available or not available. For example, a camera can be used by a single activity and, therefore, is considered an atomic resource. In contrast, aggregate resources, such as solar array power, can be used by several tasks at a time and each task can use a different amount of solar power. This makes it more difficult to represent and search for reservations.

Figure 3 shows the design of ARM. For each resource, ARM maintains a timeline that keeps track of when the resource is in use, along with the task and the priority of the task that is using it. For each reservation on this timeline,

```
Request (priority, startTime, endTime, duration)

T = resource timeline
T' = working copy of T between startTime
     and (endTime + duration)
p = -INFINITY
While p < priority
  Remove from T' all reservations with priority p
  i = earliest free interval in T' with size >=
duration
  If i exists:
    Discard from T any reservations during interval i
    Create new reservation for interval i
    Return i
  p = lowest priority in T'
Return failure
```

Figure 4: ARM Reservation Request Algorithm

ARM keeps a ticket, which can be used by tasks to access their reservations.

Before a task can use a resource it must first make a request to ARM indicating its priority, the time interval within which it would like to start using the resource and the duration that the resource will be used. If ARM can find room, it will place a reservation on the timeline and return a ticket to the requesting task. Before using the resource the task must hold a valid ticket and claim the resource. When the task is finished with the resource or otherwise no longer needs the reservation, it sends a release to ARM, which will clear out the reservation.

Although this is the nominal behavior of the system, it is unlikely that things will go so smoothly during execution. Therefore, ARM is designed to deal with unexpected situations. Unexpected events include: a task requiring a resource sooner or later than it anticipated, a task using a resource for a shorter or longer duration than it expected and a task making a reservation during a time interval in which another task already has a reservation. All of these cases are handled by ARM. The following subsections provide more detail on how these issues are resolved. In general, our approach is to associate a task priority with each reservation. Whenever there is a conflict for a resource, the task with the higher priority wins. If the tasks have equal priority, advantage is given to the task that came earlier.

Requesting a Ticket

In keeping with our goal of avoiding preemption due to resource conflicts, the resource manager will do some amount of look ahead when processing requests from tasks. However, given uncertainty during execution, the manager cannot strictly follow these reservations and must accommodate deviations in the actual timing requirements of the activities. The resource manager will have to modify the schedule, which may involve dropping lower priority reservations or preempting the current resource holder. The resource manager will attempt to give notice to the affected activities so that they can take appropriate action.

Figure 4 shows the algorithm used for making reservations. When an activity makes a request, it provides its priority, the time interval in which it would like to begin using the resource and the duration indicating how long it

intends to use the resource. Note that the duration is independent of the time interval in which it would like to start using the resource. ARM first tries to find an existing slot during the requested time interval. If none are found, it will begin removing lower priority reservations until enough space is freed or until all the remaining reservations have equal or higher priority than the requesting task.

The reservation algorithm reveals tradeoffs that were made when designing ARM. Our objective was to provide fast response to the requesting task without disturbing existing reservations. The quickest algorithm would be to first remove all reservations with a lower priority than the requesting task and then find a free space. While fast, this could also result in the unnecessary removal of reservations. The algorithm we are using is more computationally complex. It iterates through the reservation priority levels in an attempt to remove lower priority reservations first. Although of higher complexity, this algorithm better enforces graduated priority levels and only iterates a few times in practice.

There are many ways in which we could have increased the complexity of the algorithm. Instead of removing reservations when there is no room for a new request, it might be possible to relocate them. Alternatively a single higher priority reservation may be removed to preserve numerous lower priority reservations. These approaches were not taken because (a) they would have involved computationally expensive search and (b) the benefit would be reduced by the task by the task not performing as predicted, thus forcing repeated changes to the schedule.

Our approach does not consider multiple reservations simultaneously. Some tasks may require the use of several resources at the same time, requiring concurrent free intervals to be found for each resource. More complicated situations could arise if tasks require resources at temporal offset from each other. This type of scheduling is dealt by our deliberator and not ARM.

Claiming a Resource

A task may claim the resource at any time during its reservation. This addresses the uncertainty a task may have about when it needs the resource. If it is late, it can still claim the resource. However, if it needs the resource earlier, it must request a new reservation.

If another task is still holding the resource (after its reservation period), then ARM checks the priority of the tasks. The higher priority task always wins, and ties are broken in favor of the current resource owner. This avoids a possible preemption.

Releasing a Ticket

A task can release a resource at any time, providing extra free space on the timeline for new requests. However, if the task requires the resource for longer than allotted, it may keep it until a higher priority task makes a claim.

ETQ: Execution-Time Query

Even with execution-time atomic resource management, situations will arise where a task requires a different amount of a resource or time than was scheduled. For example, adverse soil conditions may make it more difficult for a rover to dig, thus using more energy and time to complete the task.

Even with execution-time atomic resource management, we will still have situations where a task requires a different amount of a resource or time than was scheduled. For example, adverse soil conditions may make it more difficult for a rover to dig, thus using more energy and time to complete the task.

One approach to dealing with this problem is to allow the task to continue operation and use more of the resource. As resource and time updates are made, the deliberator will detect problems that this extra resource use will have on the plan. For some types of resources, this approach will be fine. If an imaging task uses extra RAM, and the scheduling functionality detects that this will cause a problem, it can decide to discard some of the collected data.

Unfortunately, other types of resources cannot be so easily replenished, and this approach could lead to catastrophic failures. If the rover uses extra energy, the scheduling functionality detects a problem too late and the energy is already gone. This could prevent the rover from completing a mission critical task such as communicating with Earth. *Sometimes it is better to ask for permission than it is to ask for forgiveness.*

To deal with this challenge, our framework supports a query system that enables the executor to ask for permission before exceeding a resource limitation. This capability provides global consideration of resource and time usage during execution. When a monitor detects that the resource will be over-subscribed, instead of just completing or failing the task, it can query the deliberator. The query indicates how much more of the resource the task would like to use. The deliberator does a quick check to determine if the new resource usage would cause any conflicts. If no conflicts result, permission is granted. As we will see in the scenario, there are situations when exceeding the resource allotment is the desired behavior. For that reason, the framework does not require that execution-time resource query be used. Instead it is left the knowledge engineer to decide which tasks should "ask for permission" or "ask for forgiveness".

Similar to the design of the execution-time resource management functionality, there are alternative designs for this query capability that would provide more functionality at higher computationally expense. The deliberator could check if the conflicts resulting from a changed resource usage could be adequately repaired, and if so, give permission to the task.

Current Status

In our current framework CASPER creates abstract command sequences and executes those sequences by translating the CASPER planning activities into TDL task-tree goal nodes, which are then further expanded by TDL. In Figure 5 we graphically depict levels of responsibility

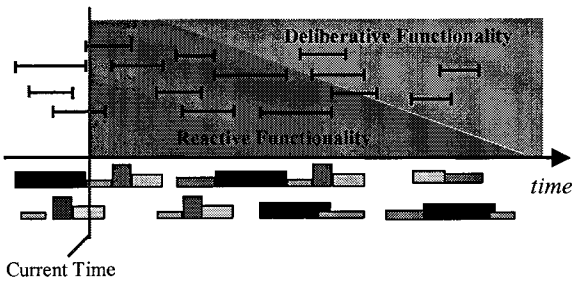

Figure 5: CLEaR Concept Diagram

between deliberative and reactive decision-making as a function of time. At the *current time*, all decision-making with respect to the executing tasks are performed reactively. As the plan is projected forward, the deliberator takes on an increasing role in decision-making.

By enabling the long-term deliberator and the short-term reactive executor to share information on a more frequent basis, the system can: (1) reduce the need to replan, (2) detect the need to replan earlier and (3) when necessary replan before entering a failed state while continuing to execute other valid portions of the plan.

Scenario Examples:

We are continuing to develop both our concept of unified planning and execution along with the implementation of that concept within the CLEaR system. To assist in this process, we are developing rover mission scenarios consistent with the proposed Mars Smart Lander mission, described in the *Mars Smart Lander Rover scenario* section, for use in testing and validating our system. We are performing tests in simulation and on the Rocky7 and Rocky8 research rovers in the JPL Mars Yard.

Figure 1 provides a high level view of the complete scenario, which includes two long-range traverses to three geological science locations and several science data gathering goals at those locations. Figure 6 contains a blow-up view of one potential geological science location. The ground operations team provides the rover with eight science targets within this site. These targets consist of: four images, two spectrometer readings, and two digs each at different locations. The ground team assigns a priority to each target, which is used in the science return optimization algorithm of the deliberator and ARM [Rabideau, et al, 2000].

Our description of the scenario will begin with events that occur while the rover is completing tasks at the geological science location. We begin with this portion of the scenario as these techniques are a logical extension to those of previous work in integrated planning and execution [Gat 1998, 1992; Bonasso, et al, 1997]. We will then move on to events that occur during the long-range traverse between science locations. During this section we will describe how these new capabilities, namely ARM and ETQ, increase the rovers ability to deal uncertain events.

We decided to turn the execution-time query facility off while the rover was in the geological science site. This was done because each of the goals in this part of the scenario is part of the rover's primary mission. If it requires extra resources to complete a task, it should do so, and the planner will have to repair the plan as best it can to achieve future goals. During the long-range traverse, in the second part of the scenario, we will use execution-time resource queries to prevent opportunistic science from interfering with the rover's primary goals.

Part 1: Within the Geological Science Location

The system begins by employing a generic Traveling Salesman Problem solver to identify an initial sequence (tour) for visiting each of the science targets. The sequence is then expanded to include all of the planner level activities required to carry out that tour. During the generation of the command sequence, all of the resource constraints are maintained. For our current scenario this means that the rover's energy and memory resource profiles must be maintained within the operations constraints. For energy this requires that the projected and actual used energy level must not drop below the prescribed margin levels. In part this is to ensure that there is enough energy available for the communications activities at the end of each day and also to ensure that there is enough energy stored in the batteries for overnight operations. For memory the system must balance the memory buffer capacity to maximize science return and ensure the availability of memory storage space for future higher priority science observations.

The dashed line in Figure 6-A indicates the initial planned sequence that the rover will take to visit the science targets. However, things will not go as planned during execution and the plan will have to be modified, as shown by the solid line in Figure 6-B. The following section highlights some of the unexpected events that occurred during execution and the challenges these events posed when coupled with the hard time and resource constraints imposed by the mission.

Deliberator

The first problem with the plan is detected before execution begins. The rover has been asked to collect more science data than it has room to store in memory. The deliberative scheduling functionality is able to detect this problem and discards low priority science targets until enough space is available for the remaining targets. In the example, image target 1 from Figure 6-A is thrown out, and a new path for visiting the remaining targets is generated.

Execution-Time Plan Manager

During execution, other resource usage issues arise. One of the challenges in execution monitoring for a system under time and resource constraints is that it is not enough to detect whether or not an action resulted in success. One must also monitor how the activity affected the rover's resources and how much time it took. For example, in Figure 6-B, the image task at target 4 and the dig at target 5

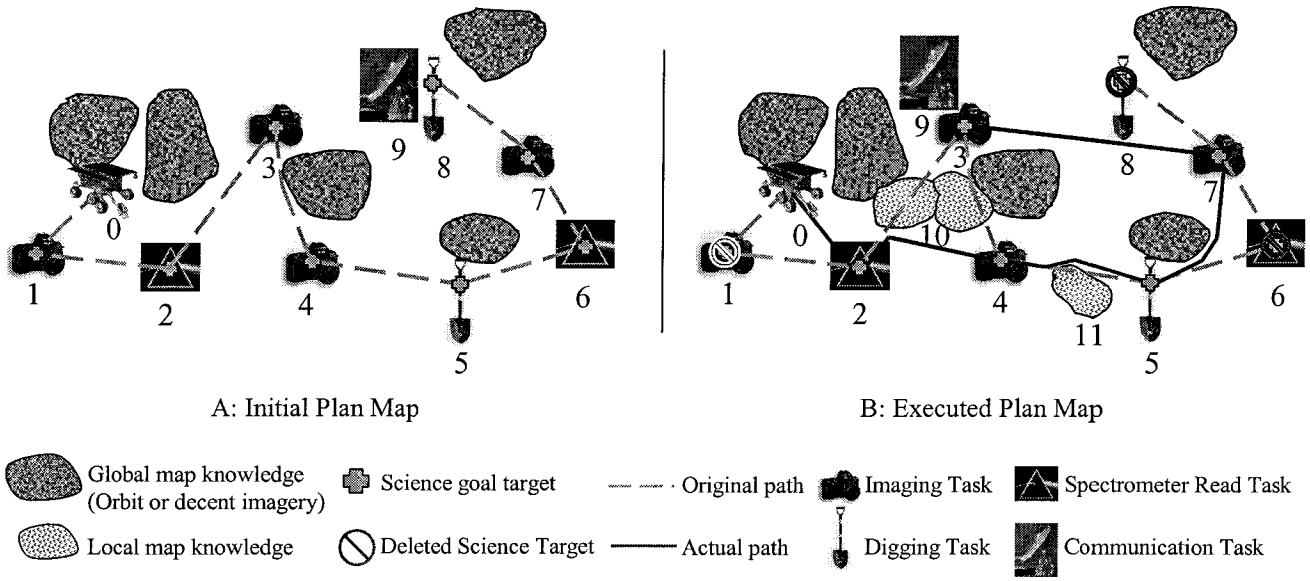

Figure 6: Scenario Maps for a Geological Science Location

were successful in that the main objective of the task was completed. However, they also resulted in the use of more resource than was anticipated. The image task required an excessive amount of memory and the dig used up too much energy.

The Execution-Time Plan Manager (Figure 2) enables the rover to deal with these problems. The Executor continuously provides updates on the state of each resource. After each task is completed, the continuous planner notices that there will be a deficiency in one or more resources. For example, after the image is taken, the system realizes that there will be insufficient memory to complete the other science goals. In each case, the deliberator looks for low priority tasks to drop, just as it did during initial plan generation.

Similar behavior occurs when a task requires an unexpected amount of time. Like the resource constraints, tight time constraints require that the rover keep track of how much time a task is taking so that it can avoid missing future deadlines. For example, as the rover moves from target 2 to target 3, its obstacle detection behavior must avoid unexpected rocks that did not show up in the initial map the rover was given. If the rover spends too much time trying to reach this target, it may miss other deadlines, such as the communication opportunity with Earth.

Again, the continuous scheduling functionality of our framework addresses this challenge. Just as each task includes monitors on resource usage, some tasks also include monitors to track the rover's progress over time. In this example, the monitor realizes that, given the rovers position, it will not be able to complete the task in the allotted time. At this point the continuous scheduling functionality takes into account the latest information about obstacles in the area and modifies the plan accordingly. As in the previous cases, it might be necessary to drop certain tasks to make up time. However, in this case, it turns out that the rover can visit the targets in a different sequence and still have enough time to make the communication deadline.

Part 2: Long-range Traverse Between Geological Science Locations

After the rover completes the tasks in Figure 6, it must proceed to the next geological science location in Figure 1. This portion of the scenario will highlight benefits of performing execution-time resource management to schedule concurrent activities that make intermittent use of the same resource. We will also show how the execution-time query facility can be used to prevent a task from interfering with a plan when it requires more of a resource.

ARM: Atomic Resource Manager

The benefits of execution-time resource management are highlighted during long-range traverses. Recall from the Mars `09 reference mission that we would like to perform opportunistic science during these traverses. Although both the traverse and traverse science tasks require the use of the camera, neither requires it continuously. These tasks can be scheduled concurrently. Due to uncertainty in execution, however, it is difficult to predict when and for how long each task will require use of the camera.

To test our execution-time resource scheduling capability, we created a simulation to model the camera usage behavior of Gestalt, the navigation software that will be used on the next Mars rover mission. Whenever Gestalt takes an image, it determines how far the rover can safely travel before it must take the next image. With an estimate of the rover's velocity, we make a prediction of when the traverse will require the camera again. A corresponding

request is made to the resource manager for the interval that the camera will be needed.

Meanwhile, our simulation of opportunistic science tries to take images as often as it can. Before using the camera, it must first make a request of ARM specifying how long it will need the camera. As stated in the '09 reference mission description, opportunistic science should not interfere with other rover activities; thus, we give opportunistic science a lower priority than traverse tasks.

Figure 7 illustrates the events that occur during a typical run. The figure depicts the reservations that are placed on the camera resource timeline during the execution of the traverse and opportunistic science tasks. Each reservation is numbered to indicate the order in which it was placed on the timeline. The time units in the x-axis are in seconds and mark the times for the various reservations. The upward arrow denotes the current point in time.

At the start of the scenario, Figure 7 (A), the traverse task has made a reservation that will begin at time 16 and last for 5 seconds, until time 21. Next, opportunistic science makes a request for the camera for sometime between second 1 and second 31. The executive resource manager finds space for the reservation starting at time 1. As time elapses, the science task completes its first use of the camera, places another request and uses the camera for a second time. At time point 12 in (C), opportunistic science requests the camera, however it cannot be given the earliest slot because its duration would conflict with traverse's reservation. Therefore, it is given a reservation that begins at time 21. In the absence of execution-time resource management, opportunistic science would take the earlier slot and later be preempted by traverse. This step demonstrates how ARM protects against preemption. Instead, opportunistic science is scheduled at a time when the resource is predicted to be free.

In (D) things do not go as planned: traverse has taken longer than expected to claim the resource. The behavior of the system at this point depends on how traverse interacts with the resource manager. If traverse releases its current reservation and makes a new one to start immediately, the resource manager will notify the following science task that it has been superseded.

However, in the example scenario, the traverse task does not release the resource and instead claims it and then holds on to it past its scheduled reservation. At time point 21, opportunistic science attempts to claim the resource but is denied in favor of the higher priority traverse. Science then makes a new request and is given a reservation starting at time slot 22. Here, no preemption was necessary to resolve the conflict, as the science task was not started.

There will be cases when preemption cannot be avoided. An example occurs in (F) when the science task has been initiated, but then the traverse requires the use of the camera beginning at time 26. Because traverse has higher priority, the resource manager gives it a reservation and preempts the opportunistic science task. (G) Shows the final state of the timeline after opportunistic science has been given a new reservation.

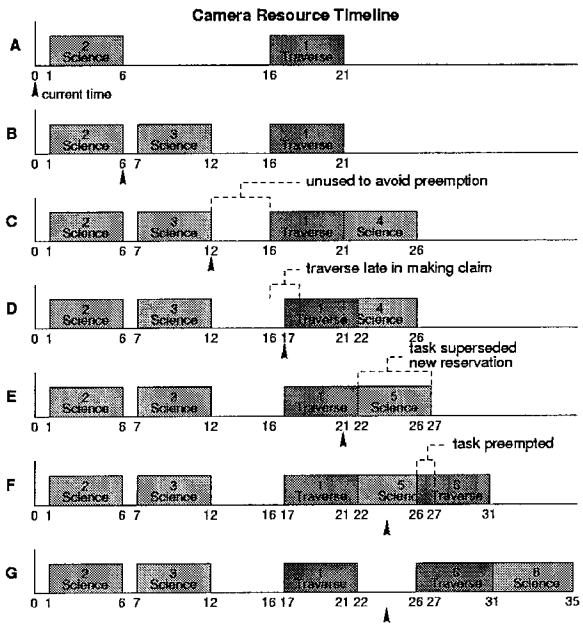

Figure 7: Example of ARM in Action

ETQ: Execution-Time Query

As stated earlier, opportunistic science should never interfere with other rover activities. Therefore, for the traverse portion of the scenario, we employ the execution-time query (ETQ) capability to enable the task manager for opportunistic science to ask permission before using more of a resource than it was prescribed by the plan.

In our scenario, the deliberator allocates a certain amount of memory for use by opportunistic science based on a rough estimate of how many images it will take and how much RAM the images will require. If, during execution, opportunistic science is able to take more images than predicted, or the images require more memory than anticipated, the task manager will detect that the task has used up the memory it has allocated. At this point, if it would like to take another image, it will use ETQ to see if it can use more memory without disrupting the plan. In its query it states the amount of additional memory it would like to use. If the scheduling facility determines that this extra usage will not cause conflicts, it will give opportunistic science the permission to take the image. However, opportunistic science will have to check again if it needs extra memory beyond this amount. If the additional use would lead to conflicts, opportunistic science would be denied and would have to stop taking images.

Related work

There have been many techniques for combining deliberative and reactive reasoning into hybrid architectures for robotic applications. These architectures have been successfully applied to many dynamic and uncertain real-world domains including manufacturing

[Lyons and Hendriks, 1995], military operations [Arkin, 1997; Myers, 1998] and space exploration [Gat, 1992; Washington, et al, 1999; Pell, et al, 1997].

[Arkin 1998] and [Knight, et al, 2001] contain surveys of many hybrid architectures. Only a few of these architectures were designed with resource constraints and tight deadlines in mind. Consequently, there has been little work in addressing these issues in dynamic, uncertain environments. Without some facility for reasoning about resources and deadlines, there is a danger that the robot will not detect problems in the plan until it is too late to do anything about it.

However, there are some architectures that are capable of reasoning about resources and deadlines. CIRCA (Musliner et al. 1993) contains a scheduler that enforces hard real-time constraints for a mobile robot navigation domains. However, rather than repair the schedule, it returns failure if it cannot meet the time constraints. CPEF [Myers, 1998] uses the SIPE-2 [Wilkins, 1988] planning system which is capable of resource management. CPEF is unique in its ability to perform indirect commanding, in which the system supervises a collection of entities executing the plan along with its ability to accept user advice for plan development. [Washington, et al, 1999] present a system that can perform resource management that is also applied to the Mars exploration domain. To deal with uncertain resource and time usage, their system precompiles resource envelopes to provide task management flexibility to the executor. The system also performs contingency planning to deal with the set of most probable plan deviations.

CLEaR extends the capabilities of the previous systems by providing execution-time atomic resource management as part of the reactive reasoning. The CLEaR framework also provides the reactive reasoning components with limited access to the global view of the plan through the execution-time query facility.

With respect to individual components of CLEaR, ARM shares some similarities with the resource manager in [Gat 2000]. In particular, both components represent execution-time resource managers. However, Gat was concerned with providing hard-real-time guarantees. In contrast, we settled for a soft-real-time system that, through the use of limited search, can do a small amount of lookahead to avoid task preemption.

Future work

Our future work will involve the use of different types of execution-time resource management, means of better utilizing path-planning algorithms in conjunction with planning and execution, means of performing quick local plan repairs while minimizing global plan risk and finding other ways of applying our unified planning and execution framework to improve mission operations, increase science return and enable more efficient long-range traverses.

We are also applying the CLEaR system to ground station automation for NASA's Deep Space Network (DSN) [DSN 1994; Fisher, et al, 1998, 2000]. In this domain area CLEaR is used in a similar fashion to generate command sequences for commanding the ground station communications subsystems to communicate with assets in deep space, whether that be Earth orbiters, spacecraft in orbit around Mars, on the surface of Mars, or as far out as Voyager I & II now beyond the edge of our solar system. The CLEaR software has also been licensed by Lockheed Martine Skunk Works division for use in automating the pilot functionality of Unmanned Air Vehicles (UAVs).

Conclusions

Resource constraints and tight deadlines pose challenges beyond those found in other uncertain robotic environments. In these applications, a task may require a different amount of time or resource than anticipated, potentially leading to execution failure at future points in the plan. We have developed the CLEaR framework to address these challenges. CLEaR extends previous work in hybrid deliberative/reactive architectures in three ways. A continuous planning and scheduling system allows the robot to identify and repair problems before they occur, while continuing to perform other tasks. ARM provides execution-time atomic resource management enabling the scheduling of concurrent tasks that require intermittent use of the same resource, while avoiding the need for task preemption. Finally, ETQ provides the executor with access to the global plan perspective needed to prevent tasks from deviating from time and resource allocations in situations when doing so will lead to conflicts. This framework will increase the effectiveness of robots in many real-world applications, including the space exploration mission presented in our case study.

Acknowledgement

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. This work has been partially supported by: the Inter-Planetary Network Information Systems Directorate (IPN-ISD) Technology Program's Mission Planning and Execution (MP&E), the Intelligent Systems' Automated Reasoning program, and the Mars Technology Program's CLARAty architecture effort. This work has also leveraged a previous large body of work including JPL's ASPEN/CASPER planning and scheduling system and CMU's Task Description Language (TDL). We would like to thank all of those involved. We would like to recognize past CLEaR team members Darren Mutz and Barbara Engelhardt. We also thank Brad Clement for his editorial comments.

References

Arkin, R., and Balch, T., 1997, AuRA: Principles and Practice in Review, *Journal of Experimental and Theoretical Artificial Intelligence,* 9(2)

Arkin, R., 1998, *Behavior-Based Robotics*, Cambridge, Massachusetts, MIT Press

Bonasso, R.P., Firby, R.J., Gat, E., Kortenkamp, D., Miller, D.P., and Slack., M.G., 1997, Experiences with an Architecture for Intelligent, Reactive Agents, In *Journal of Experimental and Theoretical Artificial Intelligence*

Chien, S., Knight, R., Stechert, A., Sherwood, R., and Rabideau, G., 2000a, Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling, In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO

Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., and Tran, D. 2000b. ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling, In *Proceedings of the SpaceOps 2000 Conference*, Toulouse, France.

(DSN, 1994) Deep Space Network, Jet Propulsion Laboratory Publication 400-517, April 1994.

Fisher, F., Knight, R., Engelhardt, B., Chien, S., and Alejandre, N., 2000, A Planning Approach to Monitor and Control for Deep Space Communications", In *Proceedings of the IEEE Aerospace Conference*

Fisher, F., Estlin, T., Mutz, D., and Chien, S., 1998, Using Artificial Intelligence Planning to Generate Antenna Tracking Plans, In *Proceedings of the Conference on Innovative Applications of Artificial Intelligence*, Orlando, Florida

Gat, E., 2000, Hard-real-time Resource Management for Autonomous Spacecraft, In *Proceedings of the 2000 IEEE Aerospace Conference.* Big Sky, MT.

Gat, E., 1998, On Three-Layer Architectures, In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, eds. *Artificial Intelligence and Mobile Robots.* MIT/AAAI Press. Cambridge, MA, 1998.

Gat, E., 1992, Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the National Conference on Artificial Intelligence*

Knight, R., Fisher, F., Estlin, T., Engelhardt, B., and Chien, S., 2001 Balancing Deliberation and Reaction, Planning and Execution for Space Robotic Applications, In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Maui, Hawaii,

Lyons, D. and Hendriks, A., 1995 Planning as incremental adaptation of a reactive system, *Robotics and Autonomous Systems*, 14(4), 255-288

Minton, S., and Johnston, M. 1988. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems." *Artificial Intelligence*, 58:161-205.

Mishkin, A., Morrison, J., Nguyen, T., Stone, H. Cooper, B., Wilcox, B., 1998, Experiences with Operations and Autonomy of the Mars Pathfinder Microrover, *Proceedings of IEEE Aerospace Conference*, Snowmass at Aspen

Musliner, D.J., Durfee, E., and Shin, K., 1993, CIRCA: A cooperative, intelligent, real-time control architecture, *IEEE Transactions on Systems, Man and Cybernetics*, 23(6)

Myers, K. L. 1998, Towards a Framework for Continuous Planning and Execution, In Proceedings of the AAAI Fall Symposium on Distributed Continual Planning

Pell, B., Gat, E., Keesing, R., Muscettola, R., and Smith, B., 1997b. Robust periodic planning and execution for autonomous spacecraft, In *Proceedings of the Int'l Joint Conference on Artificial Intelligence*

Rabideau, G., Engelhardt, B., and Chien, S. 2000. Using Generic Preferences to Incrementally Improve Plan Quality. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO.

Simmons, R. and Apfelbaum, D., 1998 A Task Description Language for Robot Control, In *Proceedings of Conference on Intelligent Robotics and Systems*, Vancouver Canada

Washington, R., Golden, K., Bresina, J., Smith, D.E., Anderson, C., and Smith, T. 1999. Autonomous Rovers for Mars Exploration. In *Proceedings of the 1999 IEEE Aerospace Conference.* Aspen, CO.

Wilkins, D. E., 1988, Causal Reasoning in Planning, *Computational Intelligence*, vol. 4, no. 4, pp. 373--380

Zweben, M., Daun, B., Davis, E., and Deale, M. 1994. Scheduling and Rescheduling with Iterative Repair, In *Intelligent Scheduling*, Morgan Kaufmann, San Francisco, CA. 241-256.