Proceedings of the 2001 IEEE/RSJ
International Conference on Intelligent Robots and Systems
Maui, Hawaii, USA, Oct. 29 - Nov. 03, 2001

Balancing Deliberation and Reaction, Planning and Execution for Space Robotic Applications

Russell Knight, Forest Fisher, Tara Estlin, Barbara Engelhardt, Steve Chien
Jet Propulsion Laboratory
California Institute of Technology
{*firstname.lastname*}@jpl.nasa.gov

Abstract

Intelligent behavior for robotic agents requires a careful balance of fast reactions and deliberate consideration of long-term ramifications. The need for this balance is particularly acute in space applications, where hostile environments demand fast reactions, and remote locations dictate careful management of consumables that cannot be replenished. However, fast reactions typically require procedural representations with limited scope and handling long-term considerations in a general fashion is often computationally expensive.

In this paper, we describe three major areas for autonomous systems for space exploration: free-flying spacecraft, planetary rovers, and ground communications stations. In each of these broad applications areas, we identify operational considerations requiring rapid response and considerations of long-term ramifications. We describe these issues in the context of ongoing efforts to deploy autonomous systems using planning and task execution systems.

1 Introduction

Recently, technologies for autonomous systems have made considerable progress. Low-level behavior control and motion, sensing, mode identification and diagnosis, task execution and control, and automated planning and resource management technologies have all made dramatic advances in both the type and size of problems that can be solved. These advances create the potential for robotic systems with the ability to operate autonomously.

However, robotic systems present unusual challenges in that they require the tight integration of multiple technologies with all of the stresses associated with a real environment (e.g., hardware unreliability, hard timing constraints, sensing uncertainty/error, etc.). This is what makes robotic systems the most powerful demonstration of intelligent agency.

The principal contributions of this paper are to: 1) identify aspects of autonomy required for autonomous systems, 2) clarify the challenges, and 3) provide examples of domains for which autonomy is useful. We believe that all of these aspects must be recognized and dealt with effectively to attain high levels of autonomy. In our discussions, we will focus on the balance between reaction and deliberation, and between planning and execution, particularly as they relate to declarative and procedural representations. Specifically, we will talk about examples of these systems in the context of three NASA robotic applications: free-flying robotic spacecraft, ground communications stations and planetary rovers.

The remainder of this paper is organized as follows. First, we define what we mean by autonomy, procedural, declarative, deliberation, and reaction. We then describe a generic architecture for planning and execution. Next, we describe how this architecture is used to tightly couple procedural and model-based reasoning as well as deliberation and reaction. We describe these concepts in the context of three NASA robotic applications. Finally we describe approaches to the problem along with considerable previous work in the area.

2 Issues in Planning and Execution

We define autonomy as the ability of a system to handle complex tasks involving environmental feedback without external intervention or supervision. Examples include the operation of free-flying robotic spacecraft for mapping and science observation missions, the operation of surface rovers on extra-terrestrial missions, and the tracking of spacecraft using ground communications stations. While there are many aspects of autonomy, we focus on issues arising from automated planning and scheduling in the context of real-time task execution. Aspects of autonomy we explore include system plan and task representation, system control, plan generation, plan execution, monitoring, and error handling.

2.1 Knowledge Representation

A complete autonomous system consists of hardware, software (engines) and models. For the purposes of this paper, we will focus on software and models. Software and models may be defined either declaratively or procedurally. Software [1] (e.g., a planning engine or task execution engine) is the control information necessary to use the information provided by models. A model is the knowledge about a domain or situation.

Procedural models represent domain knowledge by embedding it in a control stream. This is sometimes referred to as "arbitrary code." It is often quite difficult to

[1] We will assume all software is represented procedurally, although there are exceptions, e.g. Prolog.

separate the control information from the domain information in procedural models. Declarative models are static representations about events, objects, and their relationships. The software required to use a declarative model may be quite complex; this is the topic of considerable Artificial Intelligence research.

The tradeoff between procedural models and declarative models is that while procedural models can be quickly encoded for specific domains, conventional wisdom is that (compared to declarative models) procedural models are brittle and difficult to change. Declarative models do not commit to any particular control path, and thus in theory can be more flexible with respect to uncertain and unknown events. However, the software that uses these models can be slow and often requires more computational resources than software using procedural models.

2.2 Plan Generation and Execution

Plan (and/or schedule) generation is the act of devising a set of actions (the plan) to realize a task or set of goals. In order to realize a task or achieve a goal, a system may use deliberation and/or reaction. Deliberation is the process of producing a collection of executable sub-tasks that when executed result in the realization of a task/goals. This collection of executable sub-tasks is referred to as a plan. Deliberation is the search required to find an appropriate plan and typically cannot guarantee finding a solution.

Reaction is the act of producing the next executable sub-task required to achieve a task. Therefore, reactive systems generate a plan one step at a time, only generating the next step in the plan and waiting for the results before continuing. Reaction performs little or no search, and thus is able to provide a bounded time response. Reaction is a very powerful technique that deals well with real-time issues. However, reaction has the weakness that it may take harmful actions that result in suboptimal performance or even unnecessary failure because it does not perform look-ahead. Even though it makes its decisions locally without look-ahead, reaction still has many powerful capabilities such as the ability to respond to execution errors in a timely fashion through generic error handlers and task trees as well as the ability to synchronize multiple execution threads.

The tradeoff between deliberation and reaction is that deliberation may require more resources (time and computation) than reaction, resulting in missed deadlines. Reaction may waste resources trying to achieve a task, and might not solve the problems associated with achieving a task at all. Further deliberation allows for plan optimization, and the ability to solve the problem within the global context of the robotic system and/or mission desires. We see the use of both techniques to achieve robust autonomy.

Plan execution is the act of realizing a task given a plan. Technologies used for plan execution include (among others) execution software and mode identification. Execution software takes the representation of a plan and controls the hardware such that the tasks in the plan are achieved. Mode identification is the act of using sensor information and plan context to determine the state of the environment and the state of the autonomous system. The combination of execution software and mode identification allows plan execution to achieve tasks of a plan and to know that the tasks have been achieved. Likewise, failure to achieve a state can be known and reported back to the plan generation system. Reactive plan generation immediately provides a new task (probably very similar to the failed task) to be executed. If plan generation is deliberative, a new plan or revised plan is produced.

We believe that declarative and procedural representations of models as well as deliberative and reactive plan generation are required for intelligent robotic systems.

2.3 Architectural Approaches

A common approach to designing robotic automation control software is to utilize a three-tier architecture (Gat 1998) to combine different components of the system. This framework provides a template for interfacing components that operate at different levels of abstraction and provide varying time responsiveness. A three-tier architecture is typically comprised of a deliberator (planner) tier, a task executor (sequencer) tier, and a skills/controller tier.

These tiers are usually organized by the level of abstraction in which they operate. The deliberator generally produces high-level plans using AI planning techniques, which are typically computationally intensive algorithms. Thus, this tier can take a significant amount of time to respond to new updates or changes (on the order of minutes to several hours). Domain knowledge for this tier is usually encoded using a declarative model, where it can be easily utilized by different search techniques.

The executor (or sequencer) is responsible for the execution of the plans produced by the deliberator. This execution often involves further expansion of scheduled activities based on execution context (both state and task context). The executor is also responsible for monitoring activities and conditions as execution proceeds and handles exceptions as they arise. This tier must quickly react to situations, so its behavior is typically much more responsive than that of the deliberator (typically responding on the order of seconds or even tenths of seconds). Domain knowledge at this tier is usually represented using procedural constructs, which easily encode the desired behavior (e.g., looping, conditionals). In most three-tier architectures, the executor tier drives the overall system by not only dispatching commands to the controller tier but also by requesting new plans from the deliberator at the appropriate times.

The controller consists of software controllers that directly command the hardware. Each task scheduled by the executor typically corresponds to a primitive behavior defined in the controller tier for direct execution on the hardware.

3 Domains

3.1 Free-flying robotic spacecraft

One prominent domain in space exploration that requires autonomy is free-flying robotic spacecraft (Figure 1). A free-flying spacecraft remains in space until it is turned off or de-orbited. There are several aspects of operating such a spacecraft (autonomous or otherwise). These include power management, attitude control, navigation, communications, and science instrument operation. Most of these systems are already highly automated. For example, the attitude control system (ACS[2]) can be expected to manage pointing the spacecraft given an orientation.

In general, to increase the level of autonomy of free-flying spacecraft, we focus on 1) recognizing and responding to opportunities, and 2) responding to unforeseen problems. In this context, we will highlight the roles of reactive/deliberative plan generation, plan execution, and procedural/declarative model representations. It is important to note that the responsiveness of the system relies on three factors: 1) how quickly (and well) the mode can be identified, 2) how quickly (and well) a plan can be generated, and 3) how quickly (and well) the plan can be executed. Our overall goal is to make spacecraft as responsive as possible for as many different situations as possible.

We define nominal operations as spacecraft operations with no unforeseen problems and no unforeseen opportunities. Although they are highly automated, even nominal operations can require sophisticated projection and mode identification systems. Interactions between the various systems can lead to complex forecasting issues. For example, commanding the ACS to point a science instrument at a star might require power, which in turn results in commanding the ACS to point the solar arrays at the sun. But, if the system always remains in a nominal operating scenario, then we can pre-compile our plans and send them along with the spacecraft.

Continuous nominal operations rarely occur. One reason is that often our goal in flying such spacecraft is to achieve as much science as possible. If we could predict the science, we would have little need to fly most of our spacecraft. Scientific opportunities are often discovered based on the information obtained by the instruments. Historically, information is analyzed on the ground and then new science goals are sent to the spacecraft to capitalize on these opportunities. If light travel time is large or there is a communications bottleneck, this can lead to lost opportunities.

[2] For our discussion we will assume the use of propellant and thrusters to change or maintain the orientation of the spacecraft. Propellant fuels thrusters (through burning or release under pressure) and provides the mass to push against so that the spacecraft can maneuver in a vacuum.

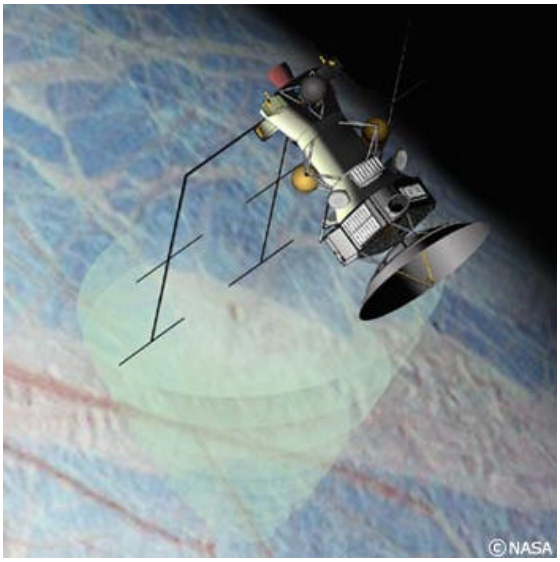

Figure 1: Europa Orbiter

Another reason continuous nominal operations rarely occur is that the environment and spacecraft hardware can be unpredictable. During off-nominal operations (where unforeseen events occur) spacecraft often do not have the technology to characterize the situation and respond accordingly. For current missions off-nominal operations often lead to the spacecraft going into a *safe mode* that ensures communications and hardware operability. The spacecraft then waits for operators on the ground to respond and direct it accordingly. Again, if light travel time is large or there is a communications bottleneck, this can lead to lost science due to waiting. Even worse, if there is no truly safe configuration that can be pre-packaged and carried along, then the spacecraft might be lost. For example, because of the time critical nature of orbit insertion, no *safe mode* exists.

Opportunities manifest themselves in a number of ways. We focus on 1) extra resource availability and 2) new science opportunities. Extra resource availability occurs when fewer resources were consumed during nominal operations. For example, the ACS system might have used less propellant than previously projected. The mode identification system must be sophisticated enough to realize that extra resources are available. This can be as simple as taking a measurement, e.g., identifying that extra propellant is available. On the other hand, identifying that extra time is available before re-entry can be a complex combination of measurements and computations. Both of these examples are traditionally modeled procedurally. A more difficult task is identifying interesting scientific events. Vision systems capable of performing such tasks require large amounts of computational resources. Once the opportunity is identified, a new plan must be generated. If a few simple steps are required that do not contend with other on-going tasks, then the system can reactively generate a

plan that takes advantage of the opportunity. On the other hand, most systems and operations interact, and the ramifications of actions on the system as a whole must be considered. If there are many interactions and many options for achieving goals, then it might be difficult or impossible to respond reactively. Deliberation provides a plan under these circumstances.

Unforeseen problems manifest themselves in a number of ways, as well. We focus on 1) resource over-utilization, 2) failed tasks, and 3) hardware or subsystem failure. Resource over-utilization occurs when a task requires more resources than expected operations. For example, the ACS system might have used more propellant than previously projected. Again, the mode identification system must be sophisticated enough to know that extra resources were used. Depending on the complexity of the projection, this may be nearly instantaneous or take a considerable amount of time and resources. Once identified, a new or modified plan must be generated that achieves the assigned tasks (or removes lower priority tasks). Failed tasks are similar in nature to over-subscriptions. The task must be identified as having failed and a new or modified plan must be generated that achieves the failed task. Hardware or subsystem failure identification is challenging. Often, hardware and subsystems fail in unpredicted ways. Mode identification under these circumstances includes sensor data, statistical inference, and model modification. Failure of a system can degrade operations (e.g., failure of a bi-directional filter wheel to turn counter-clockwise), or preclude them operating all together (e.g., failure of a communications antenna). Ideally, mode identification would characterize the changes required to the various models, but currently most models that handle system failure do so explicitly.

3.2 Deep Space Communication

Another domain area that benefits from autonomy software is NASA's Deep Space Network (DSN). The DSN (DSN 94) is the most sensitive scientific telecommunications and radio navigation network in the world (Figure 2). The purpose of the DSN is to support robotic interplanetary spacecraft missions and support radio and radar astronomy observations in the exploration of the solar system and the universe. The functions of the DSN are to receive telemetry signals from spacecraft, transmit commands that control the spacecraft operating modes, and generate the radio navigation data used to locate and guide the spacecraft to its destination. In addition, the DSN also collects flight radio science, radio and radar astronomy, very long baseline interferometry, and geo-dynamics measurements.

Current manual operations of the DSN (under nominal conditions) involves the issuing of hundreds (~700) of command directives whose selection, format, parameterization, and sequencing are dependent on the class of communication service being performed (>100 possible combinations), the instance of the equipment set,

Figure 2: 70-Meter Deep Space Communication Antenna

and type of the sub-systems used to fulfill the service (>400 combinations). For instance, different versions of a same subsystem will have variations on its command sets and on the format, type, and number of parameters. An example of how equipment assignments result in changes to the command sequence is the utilization of the 34-Meter Beam Wave Guide (BWG) antenna. This requires a different type of antenna controller than if a 34M High Efficiency (HEF) or 70M antenna is assigned to the service. The resultant antenna controller dictates both the command set (i.e., available commands and format) and the operations procedure (i.e., the necessary steps to configure and command) for that station. Planning and execution technology can be used to autonomously generate these antenna-command sequences and to handle the resolution of the above stated interactions.

For nominal operations, a DSN ground station command sequence provides the following services: uplink of spacecraft commands, downlink of telemetry[3] data and science data, handling of bit rate changes (speed of data transmission), ranging (method used to determine exact spacecraft location), and a number of others services.

The first step in the process is to configure all of the appropriate subsystems, taking into account any inter-dependencies. Using a declarative representation an autonomous system can reason in a general fashion about such interdependencies. An example of such a dependency is that configuring the receiver (RCV) requires loading of configuration tables and ensuring that the data path from

[3] Telemetry data refers to spacecraft engineering and health data representing the state of the spacecraft in conjunction with subsystem commands and responses.

the horn to the appropriate low noise amplifier is established. Another example is that the routing of the signal path is managed by the microwave controller subsystem (UWC), resulting in the ordering constraint that the UWC be configured prior to the RCV.

Due to the nature of the complex communication process carried out by the DSN it is not uncommon for failures to occur. This results in the need for recovery steps to be carried out quickly. In an autonomous system, failures are typically handled in a reactive manner. In the case of equipment failure, a simple set of local changes to the executing plan can often handle the failure. And in some causes, further global reasoning may be required to completely recover the plan from a far-reaching failure.

One example of a failure is the lack of a response from a command directive due to general unreliability in the DSN subsystems. In this case, it is often appropriate to resend the command, which is a step easily added through a reactive executive. The reason behind a retry is that it is not uncommon for the communication to the subsystems or internal to the subsystems to be temporarily interrupted. If the retry does not result in the desired effect, then it is possible that the subsystem will require resetting. In this case the appropriate behavior is for the subsystem to be immediately reset. This in turn would result in the subsystem not being configured, requiring the selection of the appropriate reconfiguration steps for that subsystem. This may result in reconfiguring other subsystems as well. Since this has global effects on the state of the system, it is usually resolved through deliberation.

An autonomous system is expected to respond to unexpected changes in system state or the required service request. For example, if an analysis of telemetry data indicates that the spacecraft is in *safe mode*, an autonomous system must respond immediately to the resulting change in service request. In this event the communications station must be able to alter its planned sequence of activities to accommodate the new/modified request. As a specific example, Mars Global Surveyor (MGS) entered *safe mode* in response to anomalies on the spacecraft. This occurrence first resulted in deviations (lower bit-rates) to the original spacecraft communications plan and subsequently resulted in the mission changing (in real-time) the request for service in an attempt to diagnose the cause of the onboard troubles. This sort of scenario further emphasizes the need for an autonomous system to provide the flexibility to receive new goals and to rapidly replan accordingly. A similar scenario is if a station is preempted it must be able to switch to providing service in a rapid fashion.

Another requirement on the part of the autonomous system is the overlapping of one communication pass with another to take advantage the setup/configuration steps. In the current paradigm of operations, each communication pass (i.e., service provided to a single mission), is treated as an isolated event consisting of: 1) a setup phase referred to as pre-track, 2) the service phase referred to as in-track, and 3) a shutdown/cleanup phase referred to as post-track. The pre-track configures the station for service from a fixed start state, and post-track returns the station to that same start state. In the manual operations of the station, the pre-track phase can take as long as 45 minutes to execute and the post track phase can take as long as 15 minutes to execute. Study of the problem has shown that much of this pre-track time could be eliminated by overlapping services from one mission to another enabling the system to utilize steps already taken by a previous communication pass. This could result in a reduction of as much as 10-50% overhead and enable increased utilization of the network. To take advantage of this synergistic interaction between the individual elements of the plan, the autonomous planning and execution system must reason at a global level.

3.3 Planetary Rovers

Autonomy not only benefits but also enables the task of coordinating rovers for in-situ planetary science (Figure 3). Several future missions are being planned to send robotic vehicles to Mars. In order to provide a high-rate of science return and to enable rovers to travel long distances, these missions will require highly autonomous rovers that require little communication with scientists and engineers on earth to perform their tasks. An autonomous rover should be able to make decisions about how to best achieve science goals as well as be able to react to its environment and handle unforeseen events while achieving these goals.

Future rovers for these missions must respond in a timely fashion to a dynamic and unpredictable environment. Rover plans must often be modified in the case of fortuitous events (such as science observations completing early) and setbacks (such as traverses taking longer than expected or hardware failures). In addition, many important science opportunities may arise suddenly, and plans handling such events cannot be created in advance. These opportunities must be handled dynamically, and often be allowed to pre-empt previously planned science activities.

Current rover operations are extremely limited and offer little space for autonomous activities. Operations for Sojourner were manually planned on the ground on a daily basis (Mishkin et al., 1998). In this mode of operations, the rover state at the start of the next day planning horizon was pre-determined based on data feedback from the previous day's operation. The science and engineering operation goals were then considered, and a plan for achieving the goals would be manually generated (which often took 5 hours or more to complete). This plan or sequence was then uplinked to the rover for execution onboard with only minimal amounts of flexibility. If an unexpected failure occurred, the rover would usually be taken into a *safe mode* by fault protection software and the rover would then wait in this state until the ground operations team could respond and determine a new plan. Correspondingly, if an unexpected fortuitous event occurred, the plan could not be

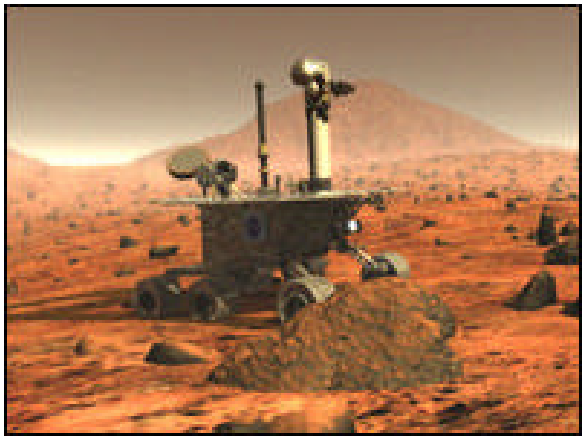

Figure 3: 2003 Mars Exploratory Rover (MER)

modified to take advantage of the situation. In our experience, there are many rover operations situations where autonomous software can provide important capabilities. For instance, providing more onboard autonomy will allow more science data to be collected and even enable new science activities, which would be virtually impossible to perform using strictly manual rover-plan generation.

One area important for rover operations is the ability to apply certain search techniques to periodically order science goals and their related traverses in an optimal (or high-quality) fashion. Before a rover plan is executed, science goals and engineering activities are often scheduled so as to reduce power resource usage. However, if something unexpected occurs during plan execution, such as an unknown obstacle blocking a pre-planned route, science goals may need to be re-ordered. For instance, assume that during a traverse a new rock formation (not seen in previous terrain imagery) is picked up by the rover's onboard sensors, and this causes the rover to go off-track in order to avoid the new obstacle. An autonomous system would recognize that the rover has gone significantly off the previously planned route and may re-order the science targets based on the rover's current location. This re-ordering often involves a significant amount of search to find a new (optimal) path, and the search process must consider the entire plan when making its decisions. This type of capability can only be performed in a deliberative fashion where search techniques can be used to find globally optimal solutions.

Flexibility in domain knowledge representation is another useful part of an autonomous rover control system. One common procedure for rover missions is to reason about different science goal priorities both before and during plan execution. A rover can only perform a limited amount of activities, and the highest priority science targets should be scheduled if at all possible. However, this schedule may have to change during execution. If a long-range traverse ends up taking much longer than previously estimated (possibly due to difficult terrain), later science goals may have to be discarded due to the limited amount of power onboard. Ideally, the least important science goal(s) should be deleted from the plan so that the more important data is still collected. This type of domain knowledge is best represented in a declarative fashion, which not only allows for more flexibility during the search process, but also enables complex goal relations and interdependencies to be represented. Also, similar to the above example, deliberative search techniques allow an autonomous system to determine the best global strategy (e.g., deleting a later science goal by evaluating overall resource consumption), where a more reactive strategy would only allow very local schedule changes.

Other rover situations require more reactive behavior. Many science opportunities need some sort of immediate reaction. For example, if a rover sensor detects a Martian dust-devil the rover will need to immediately react in order to record the event and take other relevant measurements. Even pre-planned science experiments often need reactive elements to be executed on the fly (or to fill out a partially complete sequence). For instance, a rover may have to use current sensor values to determine what exact steps to perform during an experiment. If a rover is digging or drilling, the depth the rover explores to may be dependent on the current soil consistency. Or a rover may have to evaluate a rock's material spectral readings before selecting it as a good sample to collect. Furthermore, some operations may have to be performed several times before they are successful, and the number of retries will not be known until the operation is being performed. For instance, a rover may have to try several times before grasping a rock successfully. All of these behaviors require reactive capabilities where the rover plan can be quickly adapted to current information. In addition, knowledge about these behaviors is often best represented in a procedural fashion where conditional statements and looping constructs are easy to encode.

Failures or other undesirable situations also require reactive behavior. For instance, if battery voltage suddenly drops, an autonomous system should react quickly to such an event and gracefully end the current science experiment where as much data is retained as possible. Or if the wheel slippage has caused the position estimate uncertainty to grow too large, the rover should immediately stop and perform a localization activity that would recalculate a more accurate position estimate. All of these behaviors require quick-response times and, again, are best encoded using procedural constructs.

4 Summary of Approaches

There are a number of existing systems that also integrate scheduling, planning, control, and execution monitoring. We do not attempt to review them all, but focus on some of the representative systems.

4.1 Approaches with Strong Executors

The first class of approaches to autonomy are mainly enhanced procedural executor systems, some of which are capable of more advanced deliberation than is usually credited to executors. Moreover, a few rely on declarative models and integrate minor planning or scheduling components.

Brooks' subsumption architecture (Brooks, 1986) introduced the paradigm control algorithms for autonomous robots, although it contains no hierarchy of planning, scheduling, or control. This type of architecture has often been used for mobile robot navigation, where replanning and rescheduling is a more constrained problem than in the domains described above.

TCA/TDL (Simmons 1994, Simmons, Apfelbaum 1998) is a stand-alone sequencer/controller that consists of distributed modules working with a central control module via message passing. The hierarchy that created schedules or plans in TCA operates by setting up a task tree that is expanded on the fly during execution, and simultaneously monitors feedback from the controller.

Both SOAR (Laird et al. 1987) and Guardian (Hayes-Roth, 1995) are general reasoning systems that can be adapted to a given task environment. Guardian does not have a hierarchical architecture, but uses a blackboard architecture with one module devoted to scheduling, planning, and control. SOAR also collapses all the tiers into a single reactive, rule-based mechanism.

The Spacecraft Commanding Language (SCL) (Buckley, 1991) is an executor that uses procedural scripting to execute scripts in parallel, maintain a global database, monitor changes in the database, and notify other subsystems of any changes it observes. It is designed to have very fast reaction times for critical spacecraft scenarios, and has been used aboard a number of NASA spacecraft.

CIRCA (Musliner et al. 1993) has a three-tiered architecture comprised of a planner, scheduler, and an executor that interacts with the environment through actuators and sensors in a mobile robot navigation domain. CIRCA's scheduling enforces hard real-time constraints, but it returns failure if it cannot meet the time constraints.

ATLANTIS (Gat, 1992) is comprised of a controller that acts at the lowest reactive level, a sequencer that is a special-purpose operating system based on the RAP system, and a deliberator that does planning and world modeling. In ATLANTIS, it is the sequencer that does the brunt of the work; the deliberator is under the control of the sequencer. In fact, the deliberator's output is merely used as advice by the sequencer, and the entire system is able to function without the deliberator, if necessary.

3T (Bonasso et al. 1997) is a three-tiered architecture with a planner, sequencer, and a reactive skills module that interacts with the environment. Hierarchical planning occurs before sequencing. The sequencer in 3T is a RAP (Firby, 1989) interpreter that encodes all the timing information within the RAPs. In 3T all three of its tiers do not need to be used for a given task.

AuRA (Arkin, 1989, Arkin and Balch, 1997) has three-tiers: planning, sequencing, and execution for use in mobile robot navigation. Its sequencer simply traverses a FSA expression of a plan, unlike the more powerful executives in 3T and ATLANTIS.

4.2 Formal Planning Approaches

There are a number of systems that interleave more formal planning and execution in partially observable environments using graph structures such as MDPs or formal first order logic representations. Many of these integrate execution with deliberation in order to actively gather information from the environment for the planners to generate better plans.

The RETSINA multiagent system (Paolucci, 2000) uses the HITaP planner that interleaves planning and execution in order to gather information by querying and tasking other agents and the environment. HITaP uses hierarchical task network methods to search for executable plans. When an unknown parameter is required for further decomposition, a task that will observe that parameter is executed, and the observed results are set in the BeliefDB so that planning may continue.

Alami (Alami 1998) describes a method for interleaving planning and execution for coordinating multiple robots. The core innovation is the plan merging paradigm that allows one robot to use another robot's current plan to coordinate activities and goals with its own.

Dearden (Dearden 1994) uses Markov Decision Processes (MDPs) and associated search algorithms to find near optimal plans. Search and execution are interleaved in order to gather information as the search proceeds by choosing a best action, executing that action, observing the results, and iterating. Search is performed using an envelope that focuses on nodes in the MDP relevant to the current problem.

ASPIRE (Helwig, 1996) interleaves planning and execution, using the DRIPS decision-theoretic refinement planner based on MDPs to find optimal plans. The execution module translates plan actions into commands to a physical device and monitors the progress of the actions. The search operates over the abstraction hierarchy in the planner for goals and actions.

Dervish (Nourbakhsh, 1997) interleaves conditional planning and execution with incomplete information based on simplifying assumptions and then creating a framework where incorrect assumptions still avoid unsolvable states. Assumptions are for post-conditions of certain activities that may not achieve them, and initial conditions and percepts that may be inaccurate. During run time, the assumptions are updated based on the accuracy of the predictions.

For XII (Golden, 1996), which is working on an Internet Softbot, the decision about whether to execute an activity in

the presence of incomplete information is posed as a search problem. The planner itself is an extension of UCPOP, so it can handle universal quantification and conditional effects, combined with XIIL, which is a language for representing sensory actions and information goals. One drawback of representing uncertainty related only to observation and execution is that exogenous events are not well accounted for and cannot be reasoned about thoroughly.

The IPEM system (Ambros-Ingerson, 1988) created a framework for classical planning for simple execution and replanning scenarios. It would be difficult to extend IPEM to complex planning problems, as the system does not extend to resources or real time planning, but it provides a rich framework from which many later planners could draw information and lessons.

4.3 Approaches using Strong Deliberators

The final class of approaches to autonomy include systems that focus on a powerful deliberator component and have been used on large domains.

DPLAN/NMC (Chien et al 1997) is a three-tiered architecture where the deliberator, DPLAN, is an integrated HTN and POP style planner that generated abstract antenna plans. The executive and the controller functionality were special purpose software developed for human operated antenna operations, but were later interfaced with DPLAN to perform additional automation.

DS-T (Estlin et al 1999) also utilized a traditional three-tier architecture with a batch planning system, ASPEN, and an integrated executive/software controller. The executor component comes from a COTS system, EPOCH-2000, that uses procedural scripting.

The Remote Agent Experiment (RAX), which flew on Deep Space-1, demonstrated the three tier architecture and its ability to autonomously control an active spacecraft (Pell, 1997). The HSTS batch planning system was used for the deliberator, and ESL acted as the sequencer. One problem discovered using this architecture involves including a batch planner on a reactive system. The batch planner took hours to replan after state and resource updates had invalidated the original plan, and the plan was regenerated from the original set of inputs.

CASPER (Chien et al 2000) uses a powerful integrated continuous planning and scheduling engine combined with a very simple executive. CASPER is comprised of the soft real-time version of the batch planner ASPEN by creating time windows for execution, scheduling, and planning. Although the time allocated to replanning has been reduced from minutes or hours on previous batch systems to seconds, many real time systems require even faster responses.

CLEaR (Fisher et al 2000) is a system utilizing the CASPER continuous planner in conjunction with TDL in order to provide a planning and execution system for closed loop commanding. CLEaR leverages the power of the procedural representation in TDL and the declarative representation in CASPER. The deliberative and reactive methods operate in parallel at run time to determine how to best respond to failures and take advantage of opportunities.

The CPEF system (Continuous Planning and Execution Framework) (Myers 1998), using PRS, AP, and SIPE-2, is a similar framework for integrating planning and execution. CPEF attempts to cull out key aspects of the world to monitor (as is necessary in general open-world domains). As in ASPEN, CPEF also uses iterative repair under the term “conservative repairs”. The taxonomy of failure types similarly represents action failure and re-expansion of task networks using re-decomposition.

Rogue (Haigh, 1996) uses the Prodigy4.0 planning and learning system and a number of lower level tiers (obstacle avoidance, navigation, path planning, and task planning) on Xavier the robot. The learning portion of the system maps events and execution features to the cost,and creates situation-dependent rules. The system relies on the abilities in the planner to handle recovery scenarios and may require human interaction, such as in the case of navigation, to tell the robot how far off of its estimated position it really is.

5 Conclusions

This paper has described issues in balancing deliberation and reactivity and in balancing declarative and procedural representations for autonomous robotics applications. We have discussed these issues with respect to planning and execution in the context of three NASA robotics applications: free-flying spacecraft, ground communications stations, and planetary rovers for in-situ science. We have also described some of the systems that strive to address these issues. By describing some of the challenges in these applications, we hope to encourage further research in this key area of autonomous systems.

6 Acknowledgements

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

(Alami, 1998) R. Alami, F. Ingrand, S. Qutub. "A Scheme for Coordinating Multi-robot Planning Activities and Plans Execution." In *Proceedings of European Conference on Artificial Intelligence*, Brighton UK, 1998.

(Ambrose-Ingerson, 1988) J. Ambros-Ingerson and S. Steel. “Integrating planning, execution and monitoring.” In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

(Arkin, 1989) R. Arkin. “Motor schema-based mobile robot navigation.” *International Journal of Robotics Research*, 8(4), 1989.

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

(Beetz, 1994) M. Beetz and D. McDermott. "Improving robot plans during their execution, *Proceedings of the Second International Conference on AI Planning Systems*, 1994.

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

(Brooks, 1986) R. Brooks. "A Robust Layered Control System for a Mobile Robot." In *IEEE Journal of Robotics and Automation*, 2(1), 1986.

(Buckley, 1991) Buckley, Brian and Wheatcraft, Louis: "Spacecraft Command Language - A Smart Control System, Interface and Control Systems," http://www.interfacecontrol.com, Barrios Technology, Houston, TX., March 1991.

(Chien et al, 1997) S. A. Chien, R. W. Hill Jr., A. Govindjee, X. Wang, T. Estlin, M. A. Griesel, R. Lam, and K. V. Fayyad, "A Hierarchical Architecture for Resource Allocation, Plan Execution, and Revision for Operation of a Network of Communications Antennas," *Proceeding of the IEEE Int'l Conference on Robotics and Automation* (ICRA), 1997

(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," *Proc. of the 5th Int'l Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO, April 2000.

(Dearden, 1994) R. Dearden and C. Boutilier. "Integrating planning and execution in stochastic domains." In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, Seattle, July 1994.

(Draper, 1994) D. Draper, S. Hanks, and D. Weld. "Probabilistic planning with information gathering and contingent execution." In *Proceedings of the Second Int'l Conference on Artificial Intelligence Planning Systems*, June 1994.

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

(Durfee, 1997) E.H. Durfee, P.G. Kenny, and K.C. Kluge. "Integrated permission planning and execution for unmanned ground vehicles." *In Proceedings of the First International Conference on Autonomous Agents*, 1997.

(Estlin et al 1999) T. Estlin, F. Fisher, D. Mutz, S. Chien, "Automated Planning for a Deep Space Communications Station", *Proceedings of the IEEE Conference on Robotics and Automation* (ICRA), May 1999.

(Estlin et. al. 1996) T. Estlin, X Wang, A. Govindjee, and S. Chien, "DPLAN Deep Space Network Antenna Operations Planner Programmers Guide Version 1.0," Jet Propulsion Laboratory Technical Document D-13377, February 1996.

(Firby, 1989) R.J. Firby. "Adaptive Execution in Complex Dynamic Worlds." PhD Thesis, Yale University, 1989.

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

(Gat, 1998) E. Gat. "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, 1992) E. Gat. "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots." In *Proceedings of the National Conference on Artificial Intelligence* (AAAI), 1992.

(Golden, 1996) K. Golden, O. Etzioni, and D. Weld. "Planning with execution and incomplete information." Technical report, Dept of Computer Science, University of Washington, TR96-01-09, February 1996.

(Haigh, 1996) K. Haigh, and M. Veloso. "Interleaving planning and robot execution for asynchronous user requests." In *Proceedings of the International Conference on Intelligent Robots and Systems* (IROS), 1996.

(Hayes-Roth, 1995) B. Hayes-Roth. "An Architecture for Adaptive Intelligent Systems." *Artificial Intelligence*, 72, 1995.

(Helwig, 1996) J. Helwig and P. Haddawy. "An Abstraction-Based Approach to Interleaving Planning and Execution in Partially-Obserable Domains." *AAAI Fall Symposium on Plan Execution: Problems and Issues*. November 1996.

(Laird et al. 1987) J.E. Laird, A. Newell, and P.S. Rosenbloom. "SOAR: An Architecture for General Intelligence." *Artificial Intelligence*, 33(1), 1987.

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

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

(Myers, 1998) K. L. Myers. "Towards a framework for continuous planning and execution." *Proceedings of the AAAI 1998 Fall Symposium on Distributed Continual Planning*, Menlo Park, CA, 1998.

(Nourbakhsh 1997) I. Nourbakhsh. "Interleaving Planning and Execution for Autonomous Robots." PhD thesis. Department of Computer Science, Stanford University, 1997.

(Paolucci et al, 2000) M. Paolucci, O. Shehory, and K. Sycara. "Interleaving planning and execution in a multiagent team planning environment." In CMU-RI-TR-00-01, 2000.

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

(Pemberthy & Weld 1992) J. S. Pemberthy and D. S. Weld, "UCPOP: A Sound Complete, Partial Order Planner for ADL," *Proceedings of the Third Int'l Conference on Knowledge Representation and Reasoning*, October 1992.

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

(Simmons, 1994) R. Simmons. "Structured Control for Autonomous Robots." *IEEE Transactions on Robotics and Automation*, 10(1), February 1994.

(Sycara, Pannu, 1998) K. P. Sycara and A. S. Pannu, "The RETSINA Multiagent System: Towards Integrating Planning, Execution and Information Gathering." *In Proceedings of the Second International Conference on Autonomous Agents*, 1998.