Autonomous Robotics, ??, 1?? ('??)
© ?? Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

An Autonomous Spacecraft Agent Prototype

BARNEY PELL

pell@ptolemy.arc.nasa.gov

Caelum Research Corporation, NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

DOUGLAS E. BERNARD, STEVE A. CHIEN, ERANN GAT

douglas.e.bernard@jpl.nasa.gov

Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109

NICOLA MUSCETTOLA, P. PANDURANG NAYAK

mus@ptolemy.arc.nasa.gov, nayak@ptolemy.arc.nasa.gov

Recom Technologies, NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

MICHAEL D. WAGNER

mwagner@4thplanet.com

Fourth Planet Inc., 220 Main Street, Suite 204, Los Altos, CA 94022.

BRIAN C. WILLIAMS

williams@ptolemy.arc.nasa.gov

NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

Received ??; Revised ??

Editors: ??

Abstract.

This paper describes the New Millennium Remote Agent (NMRA) architecture for autonomous spacecraft control systems. This architecture integrates traditional real-time monitoring and control with constraint-based planning and scheduling, robust multi-threaded execution, and model-based diagnosis and reconfiguration. We implemented a prototype autonomous spacecraft agent within the architecture and demonstrated the prototype in the context of a challenging autonomous mission scenario on a simulated spacecraft. As a result of this success, the integrated architecture has been selected to fly as an autonomy experiment on Deep Space One (1) S-1), the first flight of NASA's New Millennium Program (NMP), which will launch in 1998. It will be the first AI system to autonomously control an actual spacecraft.

Keywords: autonomous robots, agent architectures, action selection and planning, diagnosis, integration and coordination of multiple activities, fault protection, operations, real-time systems, modeling.

1. INTRODUCTION

The future of space exploration calls for establishing a "virtual presence" in space. This will be achieved with a large number of smart, cheap spacecraft conducting missions as ambitious as robotic rovers, balloons for extended atmospheric explorations and robotic submarines. Several new technologies need to be demonstrated to reach this goal, and one of the most crucial is on-board spacecraft autonomy.

In the traditional approach to spacecraft operations humans carry out on the ground a large number of functions including planning activities, sequencing spacecraft actions, tracking the spacecraft's internal hardware state, ensuring correct functioning, recovering in cases of failure, and subsequently working around faulty subsystems. This approach will not remain viable due to (a) round trip light time communication delays which make joysticking a deep space mission impossible and (b) a desire to limit the operations team and deep-space communications costs.

In the new model of operations, the scientists will communicate lli~}]-level science goals directly to the spacecraft. The spacecraft will then perform its own science planning and scheduling, translate those schedules into sequences, verify that they will not damage the spacecraft, and ultimately execute them without routine human intervention. In the case of error recovery, the spacecraft will have to understand the impact of the error on its previously planned sequence and then reschedule in light of the new information and potentially degraded capabilities.

To bridge the gap between the old operations model and the new one, we conducted a rapid-prototyping effort in which we demonstrated complete autonomous operations in a very challenging context: simulated insertion of a realistic spacecraft into orbit around Saturn. The mission scenario included trading off science and engineering goals and achieving the mission in the face of any single point of hardware failure. This Saturn Orbit Insertion (S01) scenario, although simplified, still contained the most important constraints and sources of complexities of a real mission, making it the most difficult challenge in the context of the most complicated mission phase of the most advanced spacecraft to date.

The unique requirements of this domain led us to the New Millennium Remote Agent (NMRA) architecture. The architecture integrates traditional real-time monitoring and control with (a) constraint-based planning and scheduling, to ensure achievement of long-term mission objectives and effectively manage allocation of scarce system resources; (b) robust multi-threaded execution, to reliably execute planned sequences 1111(1('1' conditions of uncertainty, to rapidly respond to unexpected events such as component failures, and to manage concurrent real-time activities; and (c) model-based diagnosis, to confirm successful plan execution and to infer the health of all system components based on inherently limited sensor information.

The New Millennium Remote Agent (NMRA) architecture was successfully demonstrated on the simulated SOI scenario ill October 1995. This success resulted in the inclusion of NMRA as an autonomy experiment in the first NMP mission, Deep Space 1 (1) S-1), which is scheduled to launch in mid-1998. This will be the first AI system to autonomously control an actual spacecraft

2. SCENARIO

2.1. Introduction

The Cassini Saturn Orbit Insertion (S01) was used as the scenario for developing and testing the NMRA prototype. This scenario was chosen by spacecraft engineers at JPL, because it represents one of the most challenging and well-studied problems in spacecraft operations. It entails maneuvering a complex spacecraft (Cassini) with multiply-redundant systems into orbit around Saturn, while capturing science imagery of both the rings and the planet itself and down-loading science and engineering data to the Earth. The

scenario centers around the mission-critical Main Engine burn, which slows the spacecraft to the proper velocity for achieving Saturn orbit. Any error in the start time, duration, or vector of the burn will result in mission failure. Consequently, redundant spacecraft systems (e.g. switches, gyros, and even a backup Main Engine) must be preconfigured and at the ready in case of any failure.

A simplified version of Cassini was used for modeling the prototype spacecraft, and the SOI scenario was condensed into a set of goals and constraints. An example sequence satisfying the goals and constraints was also provided by the spacecraft engineers for reference. The challenge to the autonomous system was not to duplicate this sequence, but rather to plan and execute tasks in such a manner that all constraints were satisfied. Finally, a set of guidelines was established for running the scenario and handling simulated failures.

2.2. Guidelines

The following guidelines were established for the scenario:

1. Achieve the mission goals even in the event of any single point hardware failure.
2. Consider the Saturn Orbit Insertion burn a special event that, for robustness, requires that all critical subsystems operate in their highest reliability modes.
3. Although multiple independent simultaneous failures are not considered credible, multiple sequential failures spaced far enough apart to allow recovery of one before considering the next are considered credible and must be accommodated.

2.3. Goals

The following goals define the S01 scenario:

- Use the main engine to insert the spacecraft into Saturn orbit.
- Acquire and return science images of Saturn during approach.
- Acquire and return science images of Saturn's rings near closest approach.
- Assure that the camera is protected from ring particles during ring-plane crossing.

2.4. Constraints

The models of the spacecraft as understood by the planner form the context for achieving the above goals. These models constrain the choices that the planner may make, force certain tasks to be ordered, and force the addition of tasks to allow the goals to be achieved. For the SOI scenario, the following constraints significantly affect the resulting plan:

- Available spacecraft electrical power is limited; each operating mode of each assembly requires a predefined power allocation.
- Available science data storage is limited; there is not enough room to accommodate both the Saturn approach and Saturn ring images simultaneously
- Only one spacecraft pointing direction may be commanded at a time. This couples the science imaging activity, the orbit change activity, the Earth communication activity, and the ring safety activity since all require some spacecraft axis to be pointed in a particular direction (e.g., antenna toward earth).
- A main engine burn requires several preparatory steps prior to engine ignition.

Appendix 7 provides additional details about the S01 scenario.

3. DOMAIN AND REQUIREMENTS

The spacecraft domain places a number of requirements on the software architecture that differentiates it from domains considered by other researchers. In this section, we discuss the requirements of the spacecraft domain and contrast the domain with the mobile robots (mobots) domain which has been the focus of much of the work in autonomous robotics.

3.1. Spacecraft Domain

There are three major properties of the domain that drove the architecture design.

First, a spacecraft must be able to carry on *autonomous operations for long periods of time* with no human interaction. This requirement stems from limitations of deep-space communication and the desire to cut operating expenses.

The requirement for autonomous operations over long periods is further complicated by two additional features of the domain *tight resource constraints* and *hard deadlines*. A spacecraft uses various resources, including obvious ones like fuel and electrical power, and less obvious ones like the number of times a battery can be reliably discharged and recharged. Some of these)'(,s0111'("(% are renewable, but most of them are not Hence, autonomous operation requires significant emphasis on the careful utilization of non-renewable resources and on planning for the replacement of renewable resources before they run dangerously low. Spacecraft operations are also characterized by the presence of hard deadlines, e.g., the efficiency of orbit change maneuvers is a strong function of the location of the spacecraft in its orbit, so that the time at which SOI must be achieved is constrained to lie within a two hour window. Sophisticated planning and scheduling systems are needed to meet this requirement

The second central requirement of spacecraft operation is *high reliability*. Since a spacecraft is expensive and often unique, it is essential that it achieve its mission with a high level of reliability. Part of this high reliability is achieved through the use of reliable hardware. However, the harsh environment of space and the inability to test in all flight conditions can still cause unexpected hardware failures, so the software architecture is required to compensate for such contingencies. This requirement dictates the use of an execution system with elaborate system-level fault protection capabilities. Such an executive can rapidly react to contingencies by retrying failed actions, reconfiguring spacecraft subsystems, or putting the spacecraft into a safe state to prevent further, potentially irretrievable, damage.

The requirement of high reliability is further complicated by the fact that there is *limited observability* into the spacecraft's state due to the availability of only a limited number of sensors. The addition of sensors implies added mass[1], power, cabling, and up-front engineering time and effort. Each sensor must add clear value to the mission to be justified for inclusion. Furthermore, sensors are typically no more reliable than the associated spacecraft hardware, making it that much more difficult to deduce the true state of the spacecraft hardware. These constraints dictate the use of model-based diagnosis methods for identifying the true State of the spacecraft hardware. These methods predict unobservable state variables using a spacecraft model, and can effectively handle S(,11S01" failures.

The third central requirement of spacecraft operation is that of *concurrent activity*. The spacecraft has a number of different subsystems, all of which operate concurrently. Hence, reasoning about the spacecraft 11('('(1S to reflect its concurrent nature. 111 particular, the planner/scheduler needs t () be able to schedule concurrent activities in (Iiff('writ parts of the spacecraft, including constraints between concurrent activities. The executive needs to have concurrent threads active to handle concurrent commands to different parts of the spacecraft The model-based diagnosis system needs t () handle concurrent changes in the spacecraft Stilt (, either due to scheduled events 01' due 10 failures.

3. 2. Comparison to the mobot domain

The spacecraft domain shares many important features with the mobot domain, though the similarities sometimes manifest themselves in unexpected ways. There are also some important differences, both fundamentally as well as from a more pragmatic point of view.

Both mobots and spacecraft are artifacts interacting with an unengineered environment. Both have t () be able t () deal with unexpected contingencies, deadlines, uncertainty, and limited resources when operating autonomously.

There is also an analogy between the fundamental operations performed by mobots and spacecraft A mobot's fundamental operation is to move from one position to another. The analogous operation on a spacecraft is to change its orientation. The time scales, degrees of freedom, and degrees of control are similar, though the geometry is (Iifl'('wilt. Even obstacle avoidance has an analogous feature in spacecraft attitude cont 1'01:

point ing constraints to prevent sensit ive inst ruments from pointing towards the sun. Algorithms for computing attitude trajectories are essentially path-planning algorithms in spherical coordinates. A spacecraft also moves from place to place, but t he nature of ballistic trajectories and the resource constraints of realistic spacecraft make at tit ude control a better analogy to mobot position control. A spacecraft exercises lit t le direct control over its position in space. It is limited to making very tiny adjustments to its velocity vector. The evolut ion of the velocity vector and, thus, the spacecraft's spat ial dest ination, are mostly determined when the spacecraft is designed and laun ched.

These similarities are reflected in t he st ruct ure of the NMRA architecture, which shares many features Of the canonical "till'C'C-lil~('t'" mobot cont ml architecture [2, 1 5]. There is a t op-level architectural separation between deliberat ive computations, i.e., planning and scheduling (1'S), a reactive decision-making executive (EXEC), and closed-loop real-time control (1{ '1').

Tl iere are three significant differences between the spacecraft domain and the mobot domain, each of which is manifested in a part icular architectural feat ure.

The first difference is that the source of runtime contingencies in spacecraft is usually the failure of hardware, whereas in mobots it is usually unexpected interactions with the environment. This has two important consequences for t he archit ecture.

First, in order to properly respond to hardware failures it is important to know what hardware has failed, and this is not always immediately obvious from raw sensor data (sit ice t he sensors them selves may have failed). Thus we introduce a separate tol)-level component devoted to deducing the act ual state of the hardware from observables. This is analogous to having a component dedicated to world modeling in a mobot, except that a significant part of the "world" being modeled is the spacecraft itself. In this sei ise, a spacecraft can be viewed as a mix between a mobile robot and an "immobile robot" or "immobot" as described by Williams and Nayak [43].

The second architectural result of this difference is that the executive is st ructured around a single *nominal* chain of events and any deviat ion from nominal is considered a *failure* from which it is necessary t o *recover*. This is in contrast to many mobot architectures in which all possible out comes of an action are t reated equivalent ly by t he execut ive, which usually just assesses the current sit uation wit hout prejudice regarding t he out come of t he previous action t o decide what t o (lo. This allows some mobot at chitectures to t ake advant age of serendipitous cont ingencies. It is a significant structural feat ure of current spacecraft design that unexpected contingencies are never serendipit ous, t hough t his may change in t he fu- t ure. If it does, it may require changes t o our architecture.

The second di fference bet ween mobots and spacecraft is the degree of constraint and coupling imposed by limit ed resources. Terrest rial mobots usually have enough electrical and com - puting power that these do not have to be explicit ly mai iaged. By cont rast, on a spacecraft every - thing is coupled to everything else through multiple mechanisms (e. g., power, thermal, and vibrat ional). Moreover, t he costs of spacecraft dict at c t hat all resources are ut ilitized to t he great cst extent possible, even at the cost of added complexity due to increased interactions. Thus even mundane decisions, like swit ching on a camera, have t o made in t he cont ext of the spacecraft 'S global situation. This feature manifests itself in our architecture in t wo ways. First, NMRA makes use of a concurrent, temporal planner and scheduler that can resolve potentially harmful interactions by allocating resources t o concurrent act ivit ics over specified time periods. Some mobot architectures also have planners t hat coordinat c system act ivity and resources, but many do not. Some, like 3T [2] and ATLANTIS [15] usc the planner strictly to advise t he executive, while oth ers, like Subsumpt ion [4], dispense with t he plai ner altogether. In a domain where an incorrect action can lead to mission failure far int o t he future, t he planner assumes a much greater importance. Second, many mobot archit ectures resolve act ivit y failures predominant ly by making local responses (like trying another method when the first one fails). However, as noted above, ever i switching 011 a device may have negat ive interactions wit h ot her con- current act ivities. Thus, a failure recovery sequence may 11('c(1 to be generat ed ba sed on global considerations. To address t his issue, NMRA's executive

draws on the expertise of a model-based recovery expert, in addition to the procedural knowledge encoded into traditional mobot executives. This hybrid procedural/deductive executive thus extends the strengths of mobot executives to new domains of competence.

The third difference between mobots and spacecraft is the degree of reliability and robustness that is required. The opportunities for manual intervention on a spacecraft are severely restricted when compared to a terrestrial mobile robot. Typically, the only mechanism for interacting with a spacecraft is its radio communications link. These links have fairly low bandwidths and, for deep-space missions, substantial round-trip latencies. Furthermore, a mobot can almost always safely stop where it is if it needs to perform a lengthy computation to decide on its next action. A spacecraft can almost never buy time in this way. Even during cruise when there are few externally imposed deadlines, the spacecraft's attitude control loops must still operate and be properly controlled in order to prevent sensitive instruments from pointing at the sun and keep the antenna aligned with Earth. And, of course, deadlines imposed by the spacecraft's ballistic trajectory absolutely cannot be postponed. Spacecraft also tend to cost substantially more than mobots. For these reasons, autonomy software for a spacecraft must meet a much higher standard of reliability and robustness than has been the case for *mobile robots*.

4. ARCHITECTURE OVERVIEW

Since our goal was to achieve complete autonomy for a complex domain in a limited amount of time, we chose from the outset to use a set of heterogeneous, State-of-tll(?-alt, general-purpose components that had been applied to solving specific subtasks in the domain. Hence, the main challenge was the integration Of these components. These include a temporal planner/schedule, a robust multi-threaded smart executive, and a model-based diagnosis and reconfiguration system (see figure 1).

In this section, we first, describe how the Remote Agent (RA) is embedded in the overall flight software. Then, we provide an overview of the components of the NMRA, describe the high level operational cycle, and proceed to focus in on the details of each RA component. We conclude the section with a discussion of heterogeneous knowledge representations in the RA.

4.1. Embedded Remote Agent

The relationship between the NMRA and the flight software in which it is embedded is portrayed in figure 1. When viewed as a black-box, RA sends out commands to the *real-time control* software (1{'1'). RT provides the primitive skills of the autonomous system, which take the form of discrete and continuous real-time estimation and control tasks. An example of an estimation task is the attitude determination 1001), " which notes the readings from an attitude sensor assembly (a gyroscope or a star camera) and combines it with earlier estimates to update the current estimated spacecraft attitude 11(1('. An example control task is attitude control, which uses attitude *effectors* (a set of thrusters or reaction wheels) to change the spacecraft attitude in a way to reduce the error between commanded and estimated attitude. RT responds to high-level commands by changing the mode of a control loop or state of a device and sending a message back to RA when the command has completed.

In addition, the status of all RT control loops are passed back to RA through a set of *monitors* (MON). The monitors discretize the continuous data into a set of qualitative intervals based on trends and thresholds, and pass the results back to RA. The abstraction process is fast and simple, involving discretizing a continuous variable using till'('Sllol(lS 011:111 absolute 01" relative scale. For example, the main engine temperature monitor has a fixed threshold above which it declares that the temperature is too high, while the inertial reference unit (IRU) monitor has a relative threshold tHilt measures the deviation of the observed angular acceleration from the expected angular acceleration.

4.2. RA Component Summaries

The Remote Agent itself comprises three components: a Planner/Schedule (PS), a Smart Execu-

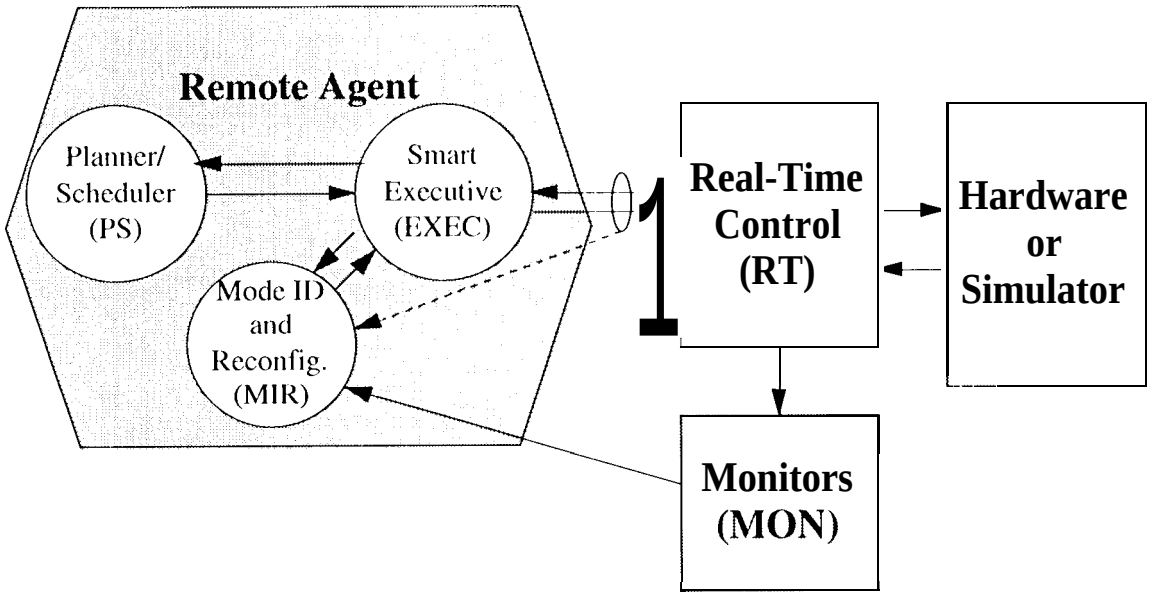

Fig. 1. NMRA architecture embedded within flight software.

tive (EXEC), and a Mode Identification and Reconfiguration component (MIR).

Planner/Scheduler (PS): PS is an integrated planner and scheduler. In our architecture, PS is activated as a "batch process" that terminates after a new schedule has been generated. It takes as input a *plan-request* that describes the current state of execution, including activities still scheduled for the future. PS combines the plan request with the goals for the current phase of the mission and produces as output a flexible, concurrent temporal plan. An output plan constrains the activity of each spacecraft subsystem over the duration of the plan, but leaves flexibility for details to be resolved during execution. The plan also contains activities and information required to monitor the progress of the plan as it is executed.

Smart Executive (EXEC): EXEC is a reactive plan execution system with responsibilities for coordinating execution-time activity.

EXEC requests a plan when necessary, by formulating a plan-request describing the current plan execution context, and then executes and monitors the generated plan. EXEC expands it's a plan by decomposing high-level activities in the plan into primitive activities, which it then executes by sending out commands, usually to the real-time control system. EXEC determines whether its commanded activities succeeded based either on direct feedback from the recipient of the command or on inferences drawn by the Mode Identification (MID) component of MIR. When some method to achieve a task fails, EXEC attempts to accomplish the task using an alternate method in that task's definition or by invoking the Mode Reconfiguration (MR) component of MIR as a *recovery expert*. If MR finds steps to restore the failing activity without interfering with other concurrent executing activities, EXEC ~)wf(n'ills those steps and then continues on with the original definition of the activity. If the EXEC is unable to execute or repair the current plan, it aborts the plan, cleans up all executing activities, and puts the controlled system into a stable safe state (called a *standby mode*). EXEC then requests a new plan while maintaining this standby mode until the plan is received, and finally executes the new plan.

Mode Identification and Reconfiguration (MIR): Like EXEC, MIR runs as a concurrent reactive process. MIR itself contains two components, one for *Mode Identification* (MID) and one for *Mode Reconfiguration* (MR). MID is responsible for providing a level of abstraction to the executive which enables EXEC to reason about spacecraft state in

terms of a set of component modes rather than a set of low-level sensor readings. In this way, our architecture separates inferential knowledge from control knowledge. MI receives information about spacecraft state from two sources. MI obtains knowledge about the *commanded* state of the system through observing every command sent by EXEC to RT. MI obtains information of the *actual* state of the system by observing the command responses sent from RT to EXEC and from the monitoring data. MI checks the commanded state against command response and monitor data, using its declarative device models, to identify the actual mode (nominal or failed) of each spacecraft component. MI sends the inferences about the most likely mode of each component to EXEC whenever the inferred mode changes. MI also sends state updates whenever EXEC has issued a command, even if nothing changed. This enables EXEC to recognize when actions fail to have any effect at all.

MR serves as a *recovery expert* to EXEC. MR takes as input a *recovery request* from EXEC. The recovery request specifies a failed activity (or a set of activities) for which EXEC desires recovery. MR maps each activity into a set of component modes that support the activity. It compares the desired component modes to the current component modes (as inferred by MI) and then (when possible) produces a *recovery plan*. The recovery plan is a sequence of operations that, when executed starting in the current state, will move the executive into a state satisfying the properties required for successful execution of the failed activity.

4.3. RA Operational Cycle

Continuous autonomous operation is achieved by the repetition of the following cycle.

1 Retrieve high level goals from the mission's goals database. In the actual mission, goals can be known at the beginning of the mission, put into the database by communication from ground mission control or can originate from the operations of spacecraft subsystems (e.g., "take more pictures of star fields to estimate position and velocity of the spacecraft").

2 Ask the *planner/scheduler* to generate a schedule. The planner receives the goals, the scheduling horizon, i.e., the time interval that the schedule needs to cover, and an initial state, i.e., the state of all relevant spacecraft subsystems at the beginning of the scheduling horizon. The resulting schedule is represented as a set of *tokens* placed on various *state variable time lines*, with temporal constraints between tokens.

3 Send the new schedule generated by the planner to the *executive*. The executive will continue executing its current schedule and start executing the new schedule when the clock reaches the beginning of the new scheduling horizon. The executive translates the abstract tokens contained in the schedule into a sequence of lower level spacecraft commands that correctly implement the tokens and the constraints between tokens. It then executes these commands, making sure that the commands succeed and either retries failed commands or generates an alternate low level command sequence that achieves the token.
In more detail, execution of a single planned activity is achieved through the following cycle:

(a) EXEC decomposes a plan-level activity into a series of primitive activities based on execution context.
(b) EXEC executes a primitive activity by sending a command to RT.
(c) RT processes the command by making a change in a control loop or device state.
(d) The monitor for the affected RT component registers the change in low-level sensor data and sends MI a new abstracted value for the state of the affected components.
(e) MI compares the command to the observations, infers the most likely actual mode of each component, and sends an update to EXEC describing the changes in any modes of interest to EXEC.
(f) EXEC compares the feedback from MI to the conditions specified in its task models to determine whether the command executed successfully. If so, it proceeds to take further steps to complete the high-level activity.

(g) If EXEC receives an update from MI indicating that an activity has failed, it tries alternative methods to achieve the activity. One such method is to invoke MR as a recovery expert. In this case, the cycle is as follow%:
   i. EXEC sends MR a recovery request to recover for any failed activities.
   ii. MR p,cll['rates a recovery plan (when possible) consistent with the current state inferred by MI and sends the plan to EXEC.
   iii. EXEC treats the recovery plan as a new method to achieve the current activity and hence proceeds to decompose it in the same manner as other activities.

4. Hard command execution failures may require the modification of the schedule in which case the executive will coordinate the actions needed to keep the spacecraft in a "safe state" and request the generation of a new schedule **fl'oil)** the planner.
5. Repeat the cycle fl'oil) step 1 when one of the following conditions apply:
   (a) Execution (real) time has reached the end of the scheduling horizon minus the estimated time needed for the planner to generate a schedule for the following scheduling horizon;
   (b) The executive has requested a new schedule as a result Of a hard failure.

We now discuss the individual components Of the RA in more detail.

4.4. Planner

The goal of the NMRA planner/scheduler **[27]** is to generate a set of synchronized high-level commands that once executed will achieve mission goals. The NMRA planner presents several features that distinguish it from other Artificial Intelligence and Operations Research approaches to the problem. In the spacecraft domain planning and scheduling aspects of the problem need to be tightly integrated. The planner 11('('(is to recursively select and schedule appropriate activities to achieve mission goals and any other subgoals generated by these activities. It also needs to synchronize activities and allocate global resources over time ((g., power and data storage capacity). In this domain (and in general) subgoals may also be generated due to limited availability of resources over time. For example, in a mission it would be preferable to keep scientific instruments on as long as possible (to maximize the amount of science gathered). However limited power availability may force a temporary instrument shut-down when other more mission critical subsystems need to be functioning. In this case the allocation of power to critical subsystems (the main result of a scheduling step) generates the subgoal "instrument must be off" (which requires the application of a planning step). Considering simultaneously the consequences of planning actions and scheduling resources enables the NMRA planner to better tune the order in which decisions are made to the characteristics of the domain and therefore can help in keeping search complexity under control. This is a significant difference with respect to classical approaches both in Artificial Intelligence and Operations Research where action planning and resource scheduling are typically addressed in two subsequent problem solving stages often by distinct software systems. Another important distinction between the NMRA planner and other classical approaches to planning is that besides activities, the planner also "schedules" the occurrence of states and conditions. Such states and conditions may need to be monitored to ensure that high level spacecraft conditions are correct for goals (such as spacecraft pointing states, spacecraft acceleration and stability requirements, etc.). '1'11('s(' States can also consume resources and have finite durations and therefore have very similar characteristics to other activities in the plan. The NMRA planner explicitly acknowledges this similarity by using a unifying conceptual primitive, the *token*, to represent both actions and states that occur over time intervals of finite extension.

The planner used in the NMRA architecture consists of a heuristic search engine that operates in the space of incomplete or partial plans [38]. Since the plans explicitly represent metric time, the planner makes use of a temporal database. As with most causal planners, 1'S begins with an incomplete plan and attempts to expand it into a complete plan by posting additional con-

straints in the database. These constraints originate from external goals and from constraint templates stored in a model of the spacecraft. The temporal database and the facilities for defining and accessing model information during search are provided by the HSTS system [26].

The domain model contains an explicit declaration of the spacecraft Sul)syst('ills on which a token will occur. In the temporal database each subsystem has an associated timeline on which the planner inserts tokens corresponding to activities and states and r('solves r('source allocation conflicts. The model also contains the declaration of duration constraints and of templates of temporal constraints between tokens, called *compatibilities*. Such constraints have to be satisfied by any schedule stored in the temporal database for it to be consistent with the physics of the domain. Temporal constraint templates serve the role of generalized planning operators and are defined for any token in the domain whether it corresponds to an activity or a state. This is a significant difference with respect to classical approaches to planning where constraint templates (also referred to as operators) are typically associated to only to actions but not States. The temporal database also provides constraint propagation services to verify the global consistency of the constraints posted so far.

The constraint template in Figure 2 describes the conditions needed for an engine burn to initiate correctly (activity *Engine_Ignition* scheduled on the *(Engine Op_State)* timeline). Constraint 5 represents a request for power thilt increases the level of *Power_Used 011* the timeline *(Power_Mgmt Power)* of an amount returned by the Lisp function call *(compute-power 'Engine_Ignition)*. Explicit invocation of external function calls provides the 111('ails for the planner to invoke "expert" modules to provide narrow but deep levels of expertise in the computation of various parameters such as durations or temperature and power levels. Access to such external knowledge is a key requirement for real-world applications of planning systems [28].

The planner operates by iteratively repairing "flaws" in the plan until all flaws are eliminated.

```
(Define-Compatibility
  ((Engine Op_State) (Engine_Ignition))
  (AND
   ;; 1.  Ignition requires good engine pressure
   (contained_by((Engine_Tanks Pressure) Good)))
   ;; 2.  Engine must have finished burn preparation
   (met_by ((Engine Op_State)(Burn_Prep)))
   ;; 3.  Engine goes into sustained burn state next
   (meets ((Engine Op_State) (Engine.Burn)))
   ;; 4. Injector temperature must be good throughout
   (contained_by((Engine_Injector Temp) Good))
   ;; 5.  Formula to determine Power consumption
   (equal ((Power_Mgmt Power)
      ( + (Lisp (compute-power 'Engine_Ignition))
          Power-Used))))
(Define_Duration_Spec
  ((Engine Op_State) (Engine-ignition))
   ;; minimum duration
   (Lisp (compute-duration 'Engine_Ignition:minimum))
   ;; maximum duration
   (Lisp (compute-duration 'Engine_Ignition:maximum)))
```

Fig. 2. Constraints on the *Engine_Burn_Ignition* activity

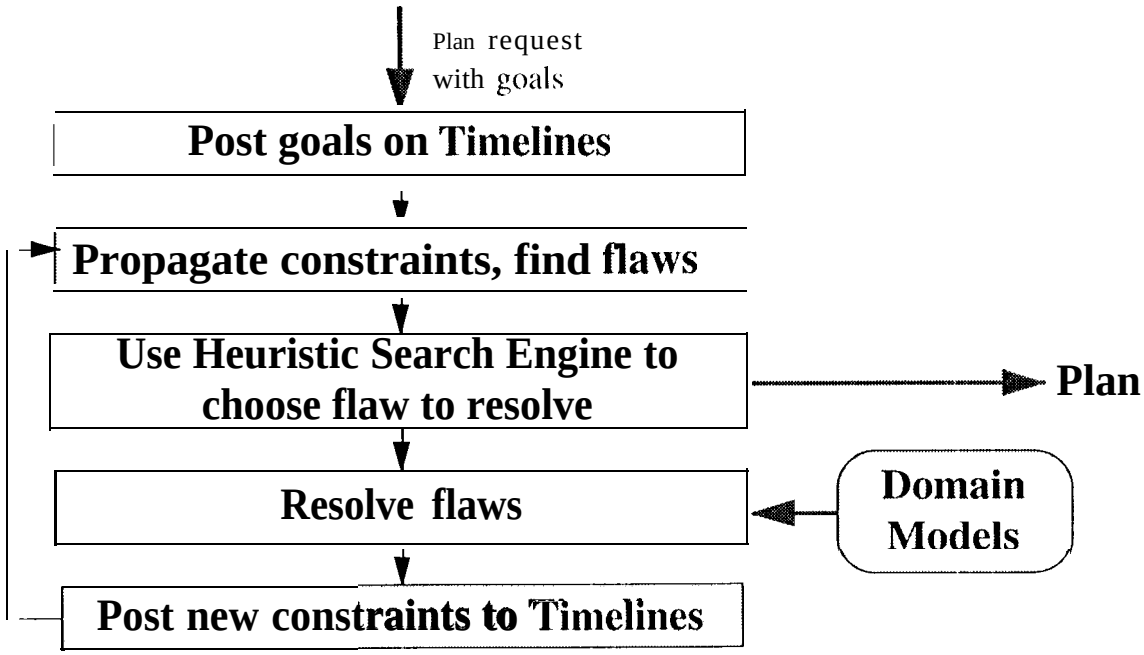

Fig. 3. Basic planner cycle

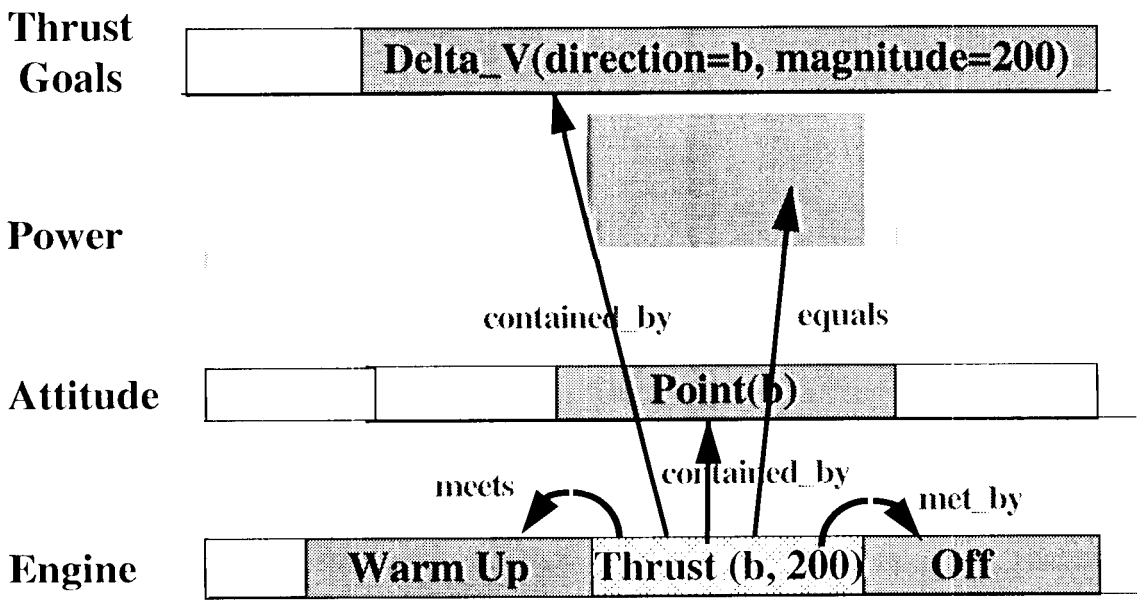

Fig. 4. A plan fragment with a completely satisfied compatibility

There are two possible kinds of flaws: 1) *unending timeline*: a plan's timeline does not end with

a token (for example, the last known token on the *(Engine Op_State)* timeline is *Engine_Idle* but the plan still allows for additional tokens to be added after it); 2) *open compatibility*: some temporal relation requested by a compatibility is still open, i.e., the planner has not selected an explicit token to satisfy it,. Figure 3 summarizes the basic "flaw fixing" loop used by the planner. To address the two possible flaw types, the planner uses the following resolution strategies. For an *unending timeline*, it takes the last token asserted on the timeline and tries to stretch it until it completely covers the rest of the timeline. If this is not possible, the temporal database will detect a temporally inconsistent plan and force backtracking. For the *open compatibility* flaw, for each temporal relation in a compatibility the planner must identify or generate a token that satisfies it. For example, Figure 4 shows a plan with the compatibility of *Thrust(b, 200)* completely satisfied, i.e., with all temporal relations associated to two tokens. If a temporal relation in a compatibility is open, the planner can use one of three resolution strategies. It can add a token to the plan in such a way that it satisfies the temporal relation; it can Select an existing token and impose the temporal relation on it; or it may notice that the needed token can fall outside of the time horizon covered by the plan and therefore decide to defer the satisfaction of the relation.

At each point in the search the planner must choose between several alternatives. Each choice is typically made using heuristics (e. g., "give highest priority among the flaws to be repaired to those associated to an *Engine_Burn* token") and, when heuristic information is not particularly strong, using a uniform randomized selection rule. If the wrong decision is made, PS will eventually reach a dead end, backtrack, and try a different path. Once the plan is free of flaws, the planner uses an iterative sampling approach [24] to heuristically improve on certain aspects of schedule quality, although it dots not guarantee even local optimality along this metric. The generation of even a single plan is costly (on the order of several CPU minutes on a SPARC20 workstation) and till(w'fore the planner needs to be called infrequently and generate plans for relatively long temporal horizons (from several hours to a week)

4.5. A Hybrid execution strategy

Runtime management of all system activities is performed by a hybrid of procedural/model-based execution capabilities [32]. The hybrid executive's functions include execution of the top-level operational cycle, plan execution, hardware reconfiguration and runtime resource management, plan monitoring, diagnosis and fault recovery. The hybrid executive invokes the planner to help it perform these functions. The executive also controls the 10 W-1 evel control software by setting its modes, supplying parameters and by responding to monitored events.

In terms of runtime management of system resources, the hybrid executive performs similar functions to a traditional operating system. The main difference is that when unexpected contingencies occur, a traditional operating system can only issue a report and abort the offending process, relying on user intervention to recover from the problem. Our executive must be able to take corrective action automatically, for example in order to meet a tight orbital insertion window.

In the event of plan failure, the executive knows how to enter a Stable State (called a Standby mode) prior to invoking the planner, and it knows how to express that standby mode in the abstract language understood by the planner. It is important to note that establishing standby modes following plan failure is a costly activity, as it causes us to interrupt the ongoing planned activities and lose important opportunities. For example, a plan failure causing us to enter standby mode during a comet encounter would cause 10ss of all the encounter science, as there is not time to re-plan before the comet is out of sight. Such concerns motivate a strong desire for plan robustness, in which the plans contain enough flexibility, and the executive has the capability, to continue execution of the plan under a wide variety of execution outcomes [33].

Our executive can be viewed as a hybrid system that shares execution responsibilities between a classical reactive execution system, RAPS []4], and a novel mode identification and reconfiguration system, called Livingstone [44]. Metaphorically, the former embodies an astronauts ability to quickly and flexibly assemble together procedural scripts into coherent control sequences, while

the later embodies an engineer's ability to reason extensively about hardware and software at a common-sense level from first principles.

Within NMRA a striking result of this hybrid was that the substantial overlap in Livingstone and RAPS ability to perform recovery and hardware configuration tasks contributed enormously to the executive's overall robustness. For example, within RAPS it was quite natural to write a set of housekeeping procedures that encode standard rules of thumb for recovery, such as "if its broken reset it" or "if its not needed, turn it off". Meanwhile, reasoning through the model allowed Livingstone to exploit the considerable redundancy in Cassini's hardware, such as identifying novel ways of exploiting partially stuck thrusters to achieve attitude adjustments. The ability of the combined system to quickly dispense with software glitches, a demonstrator's nightmare, provided a crucial turning point in the technology's acceptance.

4.6. Procedural Executive

The first half of the hybrid executive, RAPS, provides a specialized representation language for describing context-dependent contingent response procedures, with an event-driven execution semantics. The language ensures reactivity, is natural for decomposing tasks and corresponding methods, and makes it easy to express monitoring and contingent action schemas. Its runtime system then manages the reactive exploration of a space of alternative actions by searching through a space of task decompositions.

The basic runtime loop of the executive is illustrated in Figure 5. The system maintains an *agenda* on which all tasks are stored. Tasks are either active or sleeping. On each pass through the loop, the executive checks the external world to see if any new events have occured. Examples of events include model updates from the mode identification system, announcements of commanded activity completion from external software, and requests from external users. The executive responds to these events by updating its internal model of the world, changing the status of affected tasks, and installing new tasks onto the agenda. It then selects some active task (based on heuristics) and performs a small amount of processing on the task. Processing a high-level task involves breaking it up into subtasks, possibly choosing among multiple methods, whereas processing a primitive task involves sending messages to external software systems. At this point, the agenda is updated, and the basic reactive loop repeats.

RAPS encourages a close adherence to a reactive programming principle of limiting deductions within the sense-act loop to that of constructing task decompositions using a limited form of matching. This ensures quick response time, which is essential to the survival of the spacecraft. A'(vr(vtll('less it places a burden on the programmer of deducing *a priori* the consequences of failures and planning for contingencies. This is exacerbated by subtle hardware interactions, multiple and unmodeled failures, the mixture of interactions between computation, electronics and hydraulic subsystems, and limited observability due to sensor costs. These concerns are covered by the model-based component of the hybrid executive.

4.7. Mode Identification and Reconfiguration

The second half of the hybrid executive the Livingstone model-based identification and reconfiguration system (hill{) complements RAPS reactive capabilities by providing a set of deductive capabilities along the sense-act loop that operate on a single, compositional model. These models permit significant on the fly deduction of system wide interactions, used to process new sensor information (mode identification) or to evaluate the effects of alternate recovery actions (mode reconfiguration). Livingstone respects the intent of reactive systems, using propositional deductive capabilities [30] coupled to anytime algorithms that have proven exceptionally efficient in the model-based diagnosis of causal systems. Through the models, Livingstone is able to reason reactively from knowledge of failure to optimal actions that reestablish the planner's primitive goals, while mitigating the failures' effects.

Livingstone also has its limitations, which are nicely met by RAPS' procedural capabilities. Livingstone's assurance of fast inference is achieved through strong restrictions on the representation used for possible recovery actions and even more severe limitations on the way in which these ac-

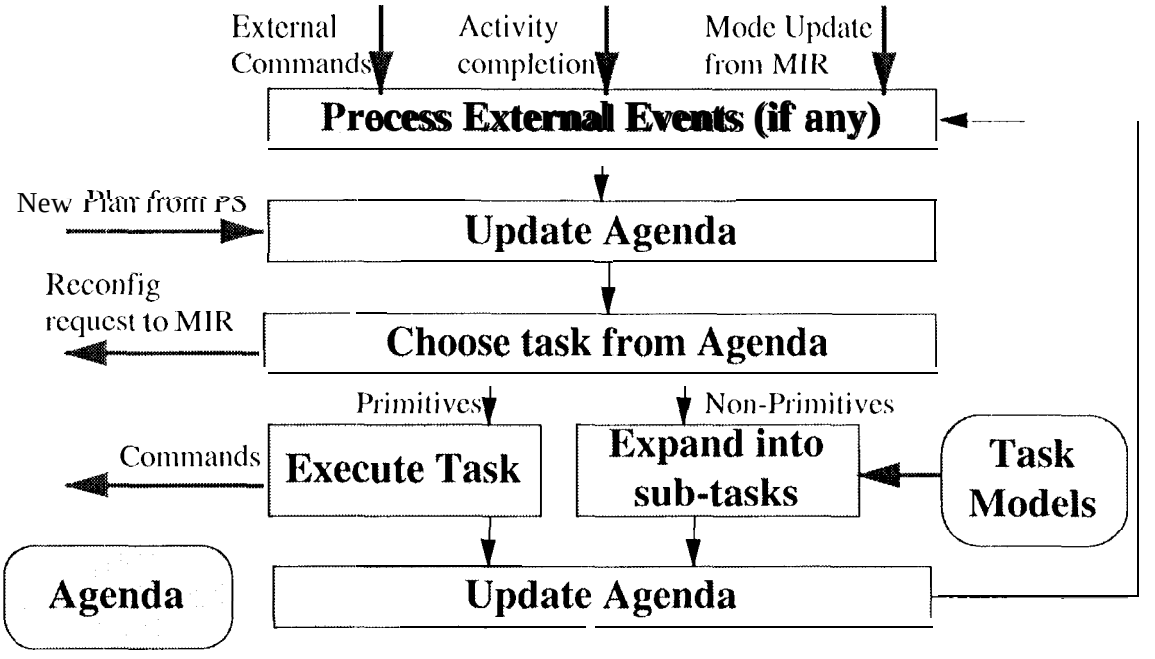

Fig. 5. Executive Task Expansion Flowchart

tions are combined (but see [45] for a much more extensive model-based execution capability that preserves reactivity). One way to preserve reactivity while improving expressiveness is for a programmer or deductive system to script these complex actions before the fact. RAPS supports this, providing a natural complement to Livingstone's deductive capabilities. For example, with respect to recovery, Livingstone provides a service for selecting, composing together and deducing the effects of basic actions, in light of failure knowledge. Meanwhile RAPS provides powerful capabilities for elaborating and interleaving these basic actions into more complex sequences, which in turn may be further evaluated through Livingstone's deductive capabilities.

Consider Livingstone in more detail. The *mode identification* (MI) component of Livingstone is responsible for identifying the current operating or failure mode of each component in the spacecraft. MI is the sensing component of Livingstone's model-based execution capability, and provides a layer of abstraction to the executive: it allows the executive to reason about the state of the spacecraft in terms of component modes, rather than in terms of low level sensor values. For example, the hybrid executive need only reason about whether a valve is open or closed, rather than have to worry about all combinations of sensor values that imply that a valve is open, and whether particular combinations of sensor values mean that the valve has failed or that a valve sensor has failed.

MI provides a variety of functions within the overall architecture. These include:

- Mode confirmation: Provide confirmation to the executive that a particular spacecraft command has completed successfully.
- Anomaly detection: Identify observed spacecraft behavior that is inconsistent with its expected behavior.
- Fault isolation and diagnosis: Identify components whose failures explain detected anomalies. In cases where models of component failure exist, identify the particular failure modes of components that explain anomalies.

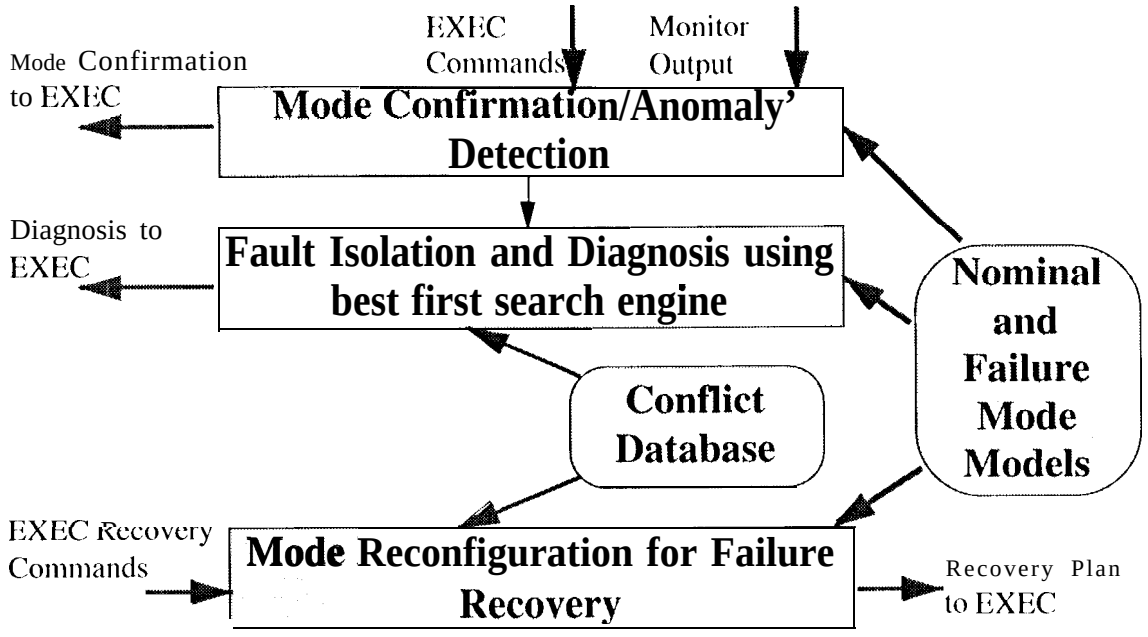

Fig. 6. Architecture of Livingstone's mode identification and reconfiguration capabilities

- Token tracking: Monitor the state of planner tokens, allowing the executive to monitor plan execution.

The *mode reconfiguration* (MR) component of Livingstone is responsible for identifying a set of control procedures that when invoked take the spacecraft from the current state, to a lowest cost state that achieves a set, of goal behaviors. MR can be used to support a variety of functions within the architecture, including:

- Mode configuration: Places the spacecraft in a least cost hardware configuration that exhibits a desired behavior.
- Recovery: Moves the spacecraft from a failure state to one that restores desired function.
- Standby and Safing. In the absence of full recovery, places the spacecraft in a safe state while awaiting additional guidance from the high-level planner or ground operations team.
- Fault avoidance: Given knowledge of current, irreparable failures, finds alternative ways of achieving desired goals.

Livingstone's MI and MR components use algorithms adapted from model-based diagnosis [8, 9] to provide the above functions (see Figure 6). The key idea underlying model-based diagnosis is that a combination of component modes is a possible description of the current state of the spacecraft only if the set of models associated with these modes is consistent with the observed sensor values. Following de Kleer & Williams [9], MI uses a conflict directed best-first search to find the most likely combination of component modes consistent with the observations. This approach is independent of the actual set of available sensors, and does not require that all aspects of the spacecraft state be directly observable, providing an elegant solution to the problem of limited observability discussed in Section 3. MR uses conflict-directed best first search to identify a least cost configuration of component modes that entail a set of goal behaviors. MR only considers those combinations that are reachable from the current state, identified by MI, through the concurrent execution of a set of component-level control procedures. This limited ability to reconfigure component modes ensures reactivity.

The use of model-based diagnosis algorithms as a foundation immediately provides Livingstone with a number of additional features. First, the search algorithms are sound and complete, pro-

viding a guarantee of coverage with respect to the models used. Second, the model building methodology is modular, which simplifies model construction and maintenance, and supports reuse. Third, the algorithms extend smoothly to handling multiple faults. Fourth, while the algorithms do not require explicit fault models for each component, they can easily exploit available fault models to find likely failures.

Livingstone extends the basic modeling paradigm used in model-based diagnosis by representing each component as a finite Stat)(' machine, and the whole spacecraft as a set of concurrent, synchronous state machines. Modeling components as finite state machines allows MI to effectively track state changes resulting from executive commands and allow's MR to plan control sequences that move from a current to target state. Modeling the spacecraft as a concurrent machine allows MI to effectively track concurrent state changes caused either by executive commands or component failures, and allows MR to plan concurrent actions.

Another important feature of Livingstone is that it models the behavior of each component mode using abstract, or qualitative, models [39, 1 O]. These abstract models are encoded as a set of propositional clauses, allowing the use of efficient incremental unit propagation for behavior prediction[30]. In addition to supporting efficient behavior prediction, abstract models are much easier to acquire than detailed quantitative engineering models, and yield more robust predictions since small changes in the underlying parameters (10 not affect the abstract behavior of the spacecraft. Spacecraft modes are a symbolic abstraction of non-discrete sensor values and are synthesized by the monitoring module.

Finally, Livingstone uses a single model and a kernel algorithm, generalized from diagnosis, to perform all of MI and MR's functions. The combination of a small kernel with a single model, and the process of exercising these through multiple uses, contributes significantly to the robustness of the complete system.

4. 8. Heterogeneous knowledge representation

One approach to developing an autonomy architecture is to seek a unified system based on a uniform computational framework. While this is an interesting goal, often the complexity of a real-world domain forces researchers to compromise 011 complete autonomy or to address simpler domains and applications. In our case the challenge was to achieve complete autonomy for a very complex domain in a limited amount of time. Therefore, we chose from the out Set to use a set of heterogeneous, state-of-the-art, general-purpose components that had been applied to solving specific subtasks in the domain, with the main challenge being the integration oft hese components. While this approach enabled us to achieve our goal of complete autonomy, it raised an important issue: the different computational engines all require different representations. These heterogeneous representations have both benefits and difficulties.

One benefit of having each engine look at the spacecraft from a different perspective is that the heterogeneous knowledge acquisition process aids in attaining *coverage and completeness*. Each new perspective on a subsystem potentially increases the understanding, and hence improves the modeling, for each of the other components which also 1 ('pt'(wit knowledge of that subsystem. Another benefit is *redundancy*, where overlapping models enable one component to compensate for restrictions in the representation of another component. This is particularly true in the hybrid executive where the rich control constructs in RAPS nicely complement the deductive capabilities of Living-Stone. A third benefit is *task specialization*, in which each component's representation can be optimized for solving a particular kind of task. This means that we can manually tailor each component's *representations* to solve problems for which it is particularly well suited.

An important example of this last point is illustrated in the representational differences between the planner/scheduler and the hybrid execution system. In NMRA the planner is concerned with activities at a high-level of abstraction which encapsulates a detailed sequence of executive-level commands. A fundamental objective for the planner is to allocate resources to the high-level activities so as to provide a time and resource enve-

lope that will ensure correctness of execution for each executive-level detailed sequence. An interval based representation of time is suitable for this purpose. From this perspective the planner does not really need to know if a time interval pertains to an activity or a state. However, this knowledge is crucial to ensure correct execution. The executive is interested in the occurrence of events, i.e., the transition between time intervals in the planner's perspective. To generate the appropriate commands and set up the appropriate sensor monitors, the executive needs to know if an event is controllable (the executive needs to send a command), observable (the executive expects sensory information), or neither (the executive can deduce information on the state on the basis of the domain model). Our approach localizes such distinctions to the executive's knowledge representation. This frees the planner to reason efficiently about intervals, and enables us to move responsibility flexibly between other architectural components (for example, let the control tasks handle an activity which was formerly decomposed by the executive, or vice-versa) without having to modify the planner's models.

While heterogeneous representations have a number of benefits, they also raise significant difficulties. Specifically, the overhead of ensuring consistency and coherence across the heterogeneous representations can be enormous. At its core, this difficulty stems from a conceptually single piece of knowledge being represented independently for each component, making it easy to introduce discrepancies. Furthermore, updating representations to reflect changes in spacecraft designs is also onerous, since the same change needs to be made (consistently) in multiple places. The traditional approach, and the primary approach we took, to solving this problem involves knowledge acquisition meetings and model review meetings involving knowledge engineering representatives from all components. However, not only are these meetings time-consuming, they are also error prone: there is rarely enough time to support in-depth reviews with all interested parties, and the resulting agreements, being in English, can lead to misunderstandings.

An alternate approach is to develop a common representation language in which each conceptually independent unit of knowledge is represented exactly once. The representations actually used by different components are automatically compiled from this single representation, thus guaranteeing consistency and coherence while retaining the benefits of heterogeneous representations discussed above. Model updates are simplified since changes need only be made in once place. We have made some progress on this front by heading toward a more unified representation of some modeled properties. First, the unified modeling for MI/MR in Livingstone (see Section 5) has proven to be extremely useful. Second, we use code generation techniques to translate some modeled properties, such as device power requirements, into the different representations used for each computational engine. We are working toward developing a single representation of the spacecraft model (the *one true model*, a holy grail of AI), by generalizing from the powerful heterogeneous models capable of handling the complexities of our real-~r(ml(] domain. We are also working on more sophisticated compilation techniques that automatically incorporate the abstractions, approximations, and reformulations needed to optimize the representation for each component.

5. IMPLEMENTATION

The implemented NMRA architecture successfully demonstrated planning of a nominal scenario, concurrent execution and monitoring, fault isolation, recovery and re-planning on a simulation of the simplified Cassini S01 scenario.

The planner modeled the domain with 22 parallel timelines and 52 (list inct temporal constraint templates. Each template included an average of 3 temporal constraints of which an average of 1.4 constraints synchronized different timelines. The resulting schedule for the nominal scenario included 200 (list inct time intervals; a schedule generated after re-planning due to engine burn interruption included 123 time intervals. The planner generated these schedules exploring less than 500" search states in an elapsed time of less than 15 minutes on a SPARC-10. Considering the computational resources available in the 1) S-1 mission and the background nature of the planning pro-

cess, this speed is acceptable with respect to the performance needed for 1)S-1.

The executive contained 100 raps with an average of 2.7 steps per rap. The nominal schedule translated into a task net with 465 steps, making it the biggest I{ AI' to date. The executive interacted with the underlying control loops which operated at a cycle frequency of 4 Hz. This performance level is higher than that needed to meet the requirements of the 1) S-1 mission.

The S01 model for the mode identification and recovery system included 80 spacecraft components with an average of 3.5 modes per component. The structure and dynamics of the domain was captured by 3424 propositions and 11101 clauses. In spite of the very large size of the model, the conflict-centered algorithms permitted fast fault isolation and determination of recovery actions. Fault isolation took between 4 and 16 search steps (1 .1 to 5.5 seconds 011 a SPARC-5) with an average of 7 steps (2.2 seconds). Recovery took between 4 and 20 steps (1.6 to 6.1 seconds) with an average of 9.3 steps (3.1 seconds).

6. RELATED WORK

The New Millennium Remote Agent (NMRA) architecture is closely related to the 3T (three-tier) architecture described in [2]. The 3T architecture consists of a deliberative component and a real-time control component connected by a reactive conditional sequencer. We and Bonasso both use RAPS [14] as our sequencer, although we are developing a new sequencer which is more closely tailored to the demands of the spacecraft environment [16].[2] Our deliberator is a traditional generative AI planner based on the HSTS planning framework [26], and our control component is a traditional spacecraft attitude control system [18]. We also add an architectural component explicitly dedicated to world modeling (the mode identifier), and distinguish between control and monitoring. In contrast to the system described by Bonasso, the prime mover in our system is the RAP sequencer, not the planner. The planner is viewed as a service invoked and controlled by the sequencer. This is necessary because computation is a limited resource (due to the hard time constraints) and so the relatively expensive operation of the planner must be carefully controlled. In this respect, our architecture follows the design of the ATLANTIS architecture[1.5].

The current state of the art in spacecraft autonomy is represented by the attitude and articulation control subsystem (A ACS) on the Cassini spacecraft [5, 1 8] (which supplied the Sol scenario used in our prototype). The autonomy capabilities of Cassini include context-dependent command handling, resource management and fault protection. Planning is a ground (rather than on-board) function and on-board replanning is limited to a couple of predefined contingencies. An extensive set of fault monitors is used to filter measurements and warn the system of both unacceptable and off-nominal behavior. Fault diagnosis and recovery are rule-based. That is, for every possible fault or set of faults, the monitor states leading to a particular diagnosis are explicitly encoded into rules. Likewise, the fault responses for each diagnosis are explicitly encoded by hand. Robustness is achieved in difficult-to-diagnose situations by setting the system to a simple, known state from which capabilities are added incrementally until full capability is achieved or the fault is unambiguously identified. The NMRA architecture uses a model-based fault diagnosis system, adds an on-board planner, and greatly enhances the capabilities of the on-board sequencer, resulting in a dramatic leap ahead in autonomy capability.

Ahmed, Aljabri, & Eldred [I] have also worked on architecture for autonomous spacecraft. Their architecture integrates planning and execution, using TCA [36] as a sequencing mechanism. However, they focused only on a subset of the problem, list of autonomous maneuver planning. Their architecture (lid not address problems of limited observability or generative planning.

Systems developed for applications other than spacecraft autonomy present S0111(" features comparable to NMRA. Bresina *et al.* [3] describe AI'A, a temporal planner and executive for the autonomous, ground-based telescope domain. Their approach uses a single action representation whereas ours uses an abstract planning language, but their plan representation shares with 11 ours flexibility and uncertainty about start and finish times of activities. However, their approach is currently restricted to single resource domains with n() con-

currency. Moreover, APA lacks a component comparable to MIR for reasoning about devices.

Phoenix [6] is an agent architecture that operates on a real-time simulated fire fighting domain. The capabilities provided by the agent are comparable to those provided by the NMRA executive although many aspects of the solution seem specific to the domain and do not appear to be easily generalizable. Unlike NMRA, Phoenix's agent does not reason explicitly about parallel action execution, since actions from instantiated plans are scheduled sequentially on a single execution timeline. A notable characteristic of Phoenix is reliance on *envelopes* [20], i.e., precomputed expected ranges of acceptability for parameters over continuous time, which are continuously monitored for robust execution.

Among the many general-purpose autonomy architectures is Guardian [21], a two-layer architecture which has been used for medical monitoring of intensive care patients. Like the spacecraft domain, intensive care has hard real-time deadlines imposed by the environment and operational criticality. One notable feature of the Guardian architecture is its ability to dynamically change the amount of computational resources being devoted to its various components. The NMRA architecture also has this ability, but the approaches are quite different. Guardian manages computational resources by changing task scheduling priorities and the rates at which messages are sent to the various parts of the system. The NMRA architecture manages computational resources by giving the executive control over deliberative processes, which are managed according to the knowledge encoded in the HAPs.

SOAR [23] is an architecture based on a general-purpose search mechanism and a learning mechanism that compiles the results of past searches for fast response in the future. SOAR has been used to control flight simulators, a domain which also has hard real-time constraints and operational criticality [37]. CIRCA [29] is an architecture that uses a slow AI component to provide guidance to a real-time scheduler that guarantees hard real-time response when possible. Noreils and Chatila [31] describe a mobile robot control architecture that combines planning, execution, monitoring, and contingency recovery.

The Cypress [40] architecture combines a planning and an execution system (SIPE-II [41] and PRS *[17])* using a common representation called ACT [42]. This serves as an example of a unified knowledge representation for use by heterogeneous architectural components, as we discussed in Section 4.8. A major difference between Cypress and NMRA is our use of an interval-based rather than an operator-based planner.

Drabble []]] describes the EXCALIBUR system, which performs closed-loop planning and execution using qualitative domain models to monitor plan execution and to generate predicted initial states for planning after execution failures. The "kitchen" domain involved concurrent temporal plans, although it was simplified and did not require robust reactions during execution.

Currie & Tate [7] describe the O-Plan planning system, which when combined with a temporal scheduler can produce rich concurrent temporal plans. Reese & Tate [35] developed art execution agent for this planner, and the combined system has been applied to a number of real-world problems including the military logistics domain. The plan repair mechanism [12] is more sophisticated then ours, although the execution agent is weaker and does not perform execution-time task decomposition or robust execution.

7. CONCLUSIONS AND FUTURE WORK

This paper has described NMRA, an implemented architecture for autonomous spacecraft. The architecture was driven by a careful analysis of the spacecraft domain, and integrates traditional real-time monitoring and control with constraint-based planning and scheduling, robust multi-threaded execution, and model-based diagnosis and reconfiguration. The implemented architecture was successfully demonstrated on an extremely challenging simulated spacecraft autonomy scenario. As a result, the architecture will fly as an experiment and control the first flight of NASA'S New Millennium Program (NMP). The spacecraft, NMP Deep Space One (DS-1), will launch in 1998 and will autonomously cruise to and fly-by an asteroid and a comet. NMRA will be the first AI

system to autonomously control an actual spacecraft

our immediate **work for** 1) S-1 consists mainly in acquiring and validating models of the 1) S-1 spacecraft and in eliciting and addressing mission requirements. To make this possible, we are working on developing better tools for sharing models across the different heterogeneous architectural components, and for model verification and validation.

Longer term, we see three major areas of research. First, our architecture could benefit from an increased use of simulation. Currently we use a simulator for development and testing the software. This could be extended to facilitate interactive knowledge acquisition and refinement, to improve projection in the planner, or to provide a tighter integration between planning and execution [1 3, 25]. Second, our architecture leaves open issues of machine learning, which could be used to tune parameters in the control system, for optimizing search control in planning, or for modifying method selection priorities during execution. Third, we see substantial benefits in having a single representation of the spacecraft, supporting multiple uses by processes of abstraction and translation. We believe that progress toward this goal is best made by generalizing from powerful, focused models capable of representing the complexities of a real-world domain.

Acknowledgements

The research described in this paper was carried out partly at the NASA Ames Research Center and partly at the Jet Propulsion Laboratory, California Institute of Technology, under contract with NASA. We would like to acknowledge the invaluable contributions of Gregg Swietek, Guy K. Man and Robert 1). Rasmussen and their tireless promotion of spacecraft autonomy. In addition to the authors, the NMRA autonomy prototype was accomplished through the efforts of Scott Davies, Charles Fry, Robert Kanefsky, Illah Nourbakhsh, Rob Sherwood, and Hans Thornas. We thank an anonymous reviewer for careful and extensive comments which improved the paper. Greg Dorais, Keith Golden, Jim Kurien, and Rich Washington provided Cotrlln('tits on drafts of this paper.

Appendix A

Saturn Orbit Insertion Scenario Details

This appendix provides an additional level of detail concerning the SOI scenario. In particular, we first describe one possible sequence of events that meets the goals and constraints of the SOI scenario. We then describe scenarios involving different failures which might happen in the course of the nominal scenario. In both cases, the sequence of events is merely representative, as actual behavior will depend on execution context.

A.0.1. Nominal Scenario

- The scenario begins one day before initial Saturn periapsis.[3]
- A plan is then generated on-board based on current on-board information about the state of the spacecraft, the spacecraft trajectory with respect to Saturn, the goals for the Saturn orbit insertion mission phase, and the system constraints.
- Ground controllers desire to know about the success of certain risky activities (such as the firing of pyrotechnic devices) early enough to take action if failures occur. This forces certain activities to be scheduled early, followed by communication of the results to the ground controllers on Earth.
- Science images are desired of Saturn initial approach and of the rings during closest approach. Limited data recorder space means that the plan should include the recorder down-load after the approach imaging and before the ring imaging.
- Power is a limited resource and engine ignition for the Sol burn occurs when power is the tightest. Nell-(ws(llt ial equipment (e. g., science instruments and reaction wheels) must be powered off prior to engine ignition.
- Some devices need to be warmed up prior to use. Each must be turned on early enough to assure availability when needed. In addition, some devices (e. g., the gyroscope and the accelerometer) must be calibrated during scheduled low activity periods before they will be available for use.

- In certain fault situations (e.g., gyro failure during main engine burn), it is imperative that a prompt switch to a backup unit be possible. otherwise the activity requiring the backup device may have to be delayed until the backup is available. If such activities are mission-critical, this could cause mission failure.
- For the critical SOI mission phase, both primary and backup units are warmed up and ready to go. The main engine is prepared for use by powering on its electronics, opening latch valves, and pre-aiming the gimbaled engine. These activities are scheduled early enough that failures allow time to switch to the backup engine.
- During SOI preparation and science collection, the spacecraft crosses the Saturn ring plane and must go to an attitude that shields the camera from ring particles.
- The spacecraft then turns to the burn attitude, main engine ignition occurs, and the spacecraft is inserted into Saturn orbit.
- After the burn, the spacecraft is returned to a safe state. Valves are closed and electronic units are powered down to clean up the state of the spacecraft and clear the way for other activities.
- The orbit insertion burn is scheduled to end at periapsis so that science observations may take advantage of the closest approach viewing.
- The ring-plane images are down-linked to the Earth as soon as possible.
- After transmission of science and engineering data to the ground, the scenario is complete.

A.0.2. Failure Scenarios

The following are examples of failure scenarios that had to be handled successfully. One class of failures involved failures in the initial phase, prior to SOI, when the main engine is being prepared for the burn. These include main engine gimbals that fail stuck when being pre-aimed, and failures of the engine valve electronics, e.g., the bipropellant latch valve driver, when being powered on. The responses to these failures involve switching to the backup main engine and preparing it for the SOI burn.

A second class of failures involved failures during the SOI burn that can be recovered from without interrupting the burn. These include failures in the inertial reference unit (IRU) and a failure of the accelerometer where it suddenly stops communicating. The response to the IRU failure is to switch to the backup IRU, which the planner has previously warmed up and readied. The response to the accelerometer communication failure is to blindly stop using accelerometer data to decide when to terminate the burn, and instead to use a timer to make the decision open-loop.

A third class of failures involved failures during the SOI burn that blind quite the burn to be interrupted. These include an engine gimbal becoming fail stuck, the main engine becoming too hot, and the accelerometer giving an acceleration reading that is blind lower than expected. In all these cases, the main engine being used for the burn is potentially unusable: an engine gimbal failure means that the engine cannot be pointed correctly to achieve SOI; an overheated main engine can irreparably damage the rest of the spacecraft; low acceleration means that the main engine is not generating enough thrust to achieve SOI. While sensor failures can also explain the above symptoms, the conservative strategy dictates assuming that the failure lies in the main engine itself. Hence, in all these cases, the response is to shut down the burn and use the backup engine to retry the burn. This requires replanning with burn restart time scheduled for when all propulsion equipment has cooled down sufficiently. The duration of this new burn must also be adjusted based on the amount of burn actually accomplished in the first attempt.

Notes

1. In a spacecraft, mass directly translates to the cost of launch and the cost of carrying extra fuel to achieve all mission maneuvers.
2. The ESL system[16] has now replaced RAPS as the core engine for the DS-1 Executive.
3. *Periapsis* refers to the point at which the spacecraft is closest to the planet.

References

1. A. Ahmed, A. S. Aljabri, and D. Eldred. Demonstration of oil-board maneuver planning using autonomous s/w architecture. In *8th Annual A/A A/USU Conference on Small Satellites*, 1991.
2. R.P. Bonasso, 1). Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *JETAI, 9(I),* 1997.
3. John Bresina, Will Edgington, Keith Swanson, and Mark 1 Drummond. Operational closed-loop observation scheduling and execution. In Pryor [34].
4. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation,* 2(1):14 23,1986.
5. G.M. Brown, D.E. Bernard, and R.D. Rasmussen. Attitude and articulation control for the cassini spacecraft: A fault tolerance overview. In *14th A IA A/IEEE Digital A vionics Systems Conference,* Cambridge, MA, Noven ber 1995.
6. P.R. Cohen, M1,. Greenberg, D. M. Hart, and A. E. 1 Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *A / Magazine,* 10(3):32 48, 1989.
7. K. Currie and A. Tate. O-plan: the open planning architecture. *Art. Int.,* 52(1):49 86, 1991.
8. Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence,* 32(1):97 130, 1987. Reprinted in [19].
9. Johan de Kleer and Brian C. Williams. Diagnosis with behavioral modes. In *Proceedings of IJCA I-89,* pages 1324 1330, 1989. Reprinted in [19].
10. Johan de Kleer and Brian C. Williams, editors. *Artificial Intelligence,* volume 51. Elsevier, 1991.
11. B. Drabble. Excalibur: A program for planning and reasoning with processes. *Artificial Intelligence,* 62(1):1 40, July 1993.
12. Brian Drabble, Austin 'Tate, and Jeff Dalton. O-plan project evaluation experiments and results. Oplan Technical Report ARPA-RL/O-Plan/TR/23 Version 1, Al AI, July 1996.
13. M. Drummond, J. Bresina, and K. Swanson. Just-in-case scheduling. In *Procs. of AA A I-94,* pages 1098 1101, Cambridge, Mass., 1994. AAAI, AAAI Press.
14. 1{. James Firby. *Adaptive execution in complex dynamic worlds.* PhD thesis, Yale University, 1978.
15. Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Procs. of AA A I-92,* Cambridge, Mass., 1992. AAA1 Press.
16. Erann Gat. ESL: A language for supporting robust plan execution in embedded autonomous agents. In Pryor *[34].*
17. Michael 1'. Georgeff and Amy L. Lansky. Procedural knowledge. Technical Report 411, Artificial Intelligence Center, SRI International, January 1987.
18. J. Hackney, D.E. Bernard, and 1{.1). Rasmussen. The cassini spacecraft: Object oriented flight control software In *1993 Guidance and Control Conference,* Keystone, CO, 1993.
19. Walter Hamscher, Luca Console, and Johan de Kleer *Readings in Model-Based Diagnosis.* Morgan Kaufmann, San Mateo, CA, 1992.
20. D. M. Hart, S. D. Anderson, and P.R. Cohen. Envelopes as a vehicle for improving the efficiency of plan execution. COINS Technical Report 90-21, Department of Computer Science, University of Massachusetts at Amherst, 1990.
21. Barbara Hayes-Roth. An architecture for adaptive intelligent systems. *Art. Int.,* 72, 1995.
22. IJCA 1. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence,* Los Altos, CA, 1997. Morgan Kaufman Publishers.
23. John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence, 33(1),* 1987.
24. P. Langley. Systematic and nonsystematic search strategies. In *Proceedings of the 1 s 1 International Conference 011 Artificial Intelligence Planning Systems,* pages 145 152. Morgan Kaufmann, 1992.
25. Richard Levinson. A general programming language for unified planning and control. *Artificial Intelligence, 76,* 1995.
26. N. Muscettola. IIS TS: Integrating planning and scheduling. In Mark Fox and Monte Zweben, editors, *Intelligent Scheduling.* Morgan Kaufmann, 199 1.
27. N. Muscettola, B. Smith, C. Chien, C. Fry, G. Rabideau, K. Rajan, and 1). Yan. On-board planning for autonomous spacecraft. In David Atkinson, editor, *Proceedings of the Fourth International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRA .$'),* Tokyo, Japan, August 1997. Jet Propulsion Laboratory.
28. Nicola Muscettola, Barney Pell, Othar Hansson, and Sunil Mohan. Automating mission scheduling for space-based observatories. In G. W. Henry and J. A. Eaton, editors, *Robotic Telescopes: Current Capabilities, Present Developments, and Future Prospects for Automated Astronomy,* number 79 in AS]' Conf. Series. Astronomical Society of the Pacific, Provo, UT, 1995.
29. David Musliner, Ed Durfee, and Kang Shin. Circa: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics,* 23(6), 1993.
30. P. Pandurang Nayak and Brian C. Williams. Fast context switching in real-time propositional reasoning. In *Procs. of AA A/-.97,* Cambridge, Mass., 1997. AA AI, AAAI Press.
31. Fabric Noreils and Raja Chatila. Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions 011 Robotics and Automation,* 1995.
32. Barney Pell, Ed Gamble, Erann Gat, Ron Keesing, Jim Kurien, hill Millar, P. Pandurang Nayak, Christian Plaunt, and Brian Williams. A hybrid procedural/deductive executive for autonomous spacecraft. In 1'. Pandurang Nayak and 1{. C. Williams, editors. *Procs. of the A A A I Fall Symposium on Model-Directed Autonomous Systems.* AAAI Press, 1997.
33. Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith. Robust peridiodic planning and execution for autonomous spacecraft. In *Procs. of IJCA* I-97 [22].
34. Louise Pryor, editor. *Proceedings of the AAAI 1,'(,11 Symposium on Plan Execution.* AAA[Press, 1996.

35. Glen Reece and Austin Tate. Synthesizing protection monitors from causal structure. In *Procs. AIPS-94*. AAAI Press, 1994.

36. Reid Simmons. An architecture for coordinating planning, sensing, and action. In *Procs. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 292–297, San Mateo, CA, 1990. DARPA, Morgan Kaufmann.

37. M. Tambe, W. Lewis Johnson, R. M. Jones, F. Koss, J. E. Laird, Paul S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1):15–39, Spring 1995.

38. D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 1994.

39. Daniel S. Weld and Johan de Kleer, editors. *Readings in Qualitative Reasoning About Physical Systems*. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

40. D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *JETAI*, 7(1):197–227, 1995.

41. David E. Wilkins. *Practical Planning*. Morgan Kaufman, San Mateo, CA, 1988.

42. David E. Wilkins and Karen L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 1995.

43. Brian C. Williams and P. Pandurang Nayak. Immobile robots: AI in the new millennium. *AI Magazine*, 17(3):16–35, 1996.

44. Brian C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Procs. of AAAI-96*, pages 971–978, Cambridge, Mass., 1996. AAAI, AAAI Press.

45. Brian C. Williams and P. Pandurang Nayak. A reactive planner for a model-based executive. In *Procs. of IJCAI-97 [22]*.

Dr. Doug Bernard received his B.S. in Mechanical Engineering and Mathematics from the University of Vermont, his M.S. in Mechanical Engineering from Cornell and his Ph.D. in Aeronautics and Astronautics from Stanford University. He has participated in dynamics analysis and attitude control system design for several spacecraft at JPL, and Hughes Aircraft, and was the Attitude and Articulation Control Subsystem (AACS) systems engineering lead for the Cassini mission to Saturn. Currently, Dr. Bernard is group supervisor for the flight system engineering group at JPL and Program Element Manager for the Remote Agent Experiment for New Millennium Program's Deep Space One mission.

Dr. Steve Chien is Technical Group Supervisor of the Artificial Intelligence Group, at the Jet Propulsion Laboratory, California Institute of Technology where he leads a multi-million dollar laboratory technology area in automated planning and scheduling for: spacecraft mission planning, maintenance of spacecraft transportation systems, science data analysis, and Deep Space Network Antenna operations. He received a Ph.D. in Computer Science in 1990 from the University of Illinois. Dr. Chien is a 1995 recipient of the Lew Allen Award for Excellence, JPL's highest award for early-career achievement. In 1997 he was awarded the NASA Exceptional Achievement Medal. Dr. Chien's research interests lie in the areas of planning, scheduling, operations research, and machine learning and he has authored numerous publications in these areas.

Dr. Erann Gat is a senior member of the technical staff at the Jet Propulsion Laboratory, California Institute of Technology, where he has been working on autonomous control architectures since 1988. In 1991 Dr. Gat developed the ATLANTIS control architecture, one of the first integrations of deliberative and reactive components to be demonstrated on a real robot. ATLANTIS was used as the basis for a robot called Alfred which won the 1993 AAAI mobile robot contest. Dr. Gat was also the principal architect of the control software for Rocky III and Rocky IV, the direct predecessors of the Pathfinder Sojourner rover. Dr. Gat escapes the dangers of everyday life in Los Angeles by pursuing safe hobbies like skiing, scuba diving, and flying small single-engine airplanes.

Dr. Nicola Muscettola is a Senior Computer Scientist at the the *Computational* Sciences Division of the NASA Ames Research Center. He received his Diploma di Laurea in Electrical and Control Engineering and his Ph.D. in Computer Science from the Politectnico di Milano, Italy. He is the principal designer of the HSTS planning framework and is the lead of the on-board planner team for the Deep Space 1 Remote Agent Experiment. His research interests include planning, scheduling, temporal reasoning, constraint propagation, action representations and knowledge compilation.

Dr. Pandurang Nayak is a *Senior Computer* Scientist at the Computational Sciences Division of the NASA Ames Research Center. He received a B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Bombay, and a Ph.D. in Computer Science from Stanford University. His Ph.D. dissertation, entitled "Automated Modeling of Physical Systems", was an ACM Distinguished Thesis. He is currently an Associate Editor of the Journal of Artificial Intelligence Research (JAIR), and his research interests include model-based autonomous systems, abstractions and approximations in knowledge representation and reasoning, diagnosis and recovery, and qualitative and causal reasoning.

Dr. Barney Pell is a Senior Computer Scientist in the Computational Sciences Division at NASA Ames Research Center. He is one of the architects of the Remote Agent for New Millennium's Deep Space One (DS-1) mission, and leads a team developing the Smart Executive component of the DS-1 Remote Agent. Dr. Pell received a B. S. degree with distinction in Symbolic Systems at Stanford University. He received a Ph.D. in computer science at Cambridge University, England, where he studied as a Marshall Scholar. His current research interests include spacecraft autonomy, integrated agent architecture, reactive execution systems, collaborative software development, and strategic reasoning. Pell was guest editor for Computational Intelligence Journal in 1996 and has given tutorials on autonomous agents, space robotics, and game-playing.

Michael D. Wagner received his BSE in Electrical Engineering from Duke University in 1989 and his MSE in Electrical Engineering/Artificial Intelligence from the University of Southern California in 1991. From 1989 until 1996, he served as an officer in the US Air Force, where he worked in the research and development of spacecraft sensor systems. His final Air Force assignment was as Telerobotics Representative to NASA Ames Research Center, where he developed SCLISOI software for the Russian Marsokhod Martian rover and led the Ames team in the development of the New Millennium Autonomy Architecture Rapid Prototype (NewMAAP). In 1996, Michael co-founded Fourth Planet, Inc., a company specializing in the visualization of complex, real-time information.

Brian C. Williams is Technical Group Supervisor of the Intelligent Autonomous Systems Group at the NASA Ames Research Center, and co-lead of the model-based autonomous systems project. He received his bachelor's in Electrical Engineering at MIT, continuing on to receive a Masters and PhD in Computer Science. While at MIT he developed one of the earliest qualitative simulation systems, TQA, a hybrid qualitative/quantitative symbolic algebra system, MINIMA, and a system IBIS for synthesizing innovative controller designs. Williams was at Xerox PARC from 1989 to 1992, where he is best known for his work on the GDE and Sherlock model-based diagnosis systems. Williams received AAAI best paper awards in 1988 and 1997 for his work on qualitative symbolic algebra and incremental truth maintenance. He was guest editor for Artificial Intelligence in 1992, Chair of the AAAI Tutorial Forum in 1996 and 1997, and is currently on the editorial board of the Journal of Artificial Intelligence Research.