Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling

Steve Chien, Russell Knight, Andre Stechert, Rob Sherwood, and Gregg Rabideau

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

Abstract

The majority of planning and scheduling research has focused on batch-oriented models of planning. This paper discusses the use of iterative repair techniques to support a continuous planning process as is appropriate for autonomous spacecraft control. This allows the plan to incorporate execution feedback - such as early or late completion of activities, and over-use or under-use of resources. In this approach, iterative repair supports continuous modification and updating of a current working plan in light of changing operating context.

Introduction

In recent years Galileo, Clementine, Mars Pathfinder, Lunar Prospector, and Cassini have all demonstrated a new range of robotic missions to explore our solar system. However, complex missions still require large teams of highly knowledgeable personnel working around the clock to generate and validate spacecraft command sequences. Increasing knowledge of our Earth, our planetary system, and our universe challenges NASA to fly large numbers of ambitious missions, while fiscal realities require doing so with budgets far smaller than in the past. In this climate, the automation of spacecraft commanding becomes an endeavor of crucial importance.

Autonomous spacecraft are made possible by equipping the spacecraft with on-board software that provides knowledge and reasoning procedures to determine appropriate actions that achieve mission goals, to monitor spacecraft health during execution, and to recover autonomously from possible faults (Muscettola et al. 1999). An on-board planner/scheduler is a key component of such a highly autonomous system.

Recent experiences indicate the promise of planning and scheduling technology for space operations. Use of the DATA-CHASER automated planning and scheduling system (DCAPS) to command the DATA-CHASER shuttle payload reduced commanding-related mission operations effort by 80% and increased science return by 40% over manually generated sequences (Chien et al. 1999). This increase was possible because short turn-around times (approximately 6 hours) imposed by operations constraints did not allow for lengthy, manual optimization. And the Remote Agent Experiment (ARC, JPL et al. 1999) demonstrated the feasibility of flying AI software (including a planner) to control a spacecraft.

This paper describes a further step in incremental planning and scheduling. In this approach, rather than constructing batch back-to back plans, a persistent agent always has a plan for a fixed time span in to the future.

The remainder of this paper is organized as follows. First, we describe our approach to interleaving planning and execution and how it improves the responsiveness of the planning component. Next, we describe technical details of our approach to interleaving planning and execution and how this approach is used to reduce this response time of the planner/scheduler. We then highlight how this system works using examples from spacecraft operations. We then describe an empirical evaluation of our approach in a stochastic domain. Finally, we describe future work and related work and conclusions.

Integrating Planning and Execution

Traditionally, much of planning and scheduling research has focused on a batch formulation of the problem. In this approach (see Figure 1), time is divided up into a number of planning horizons, each of which lasts for a significant period of time. When one nears the end of the current horizon, one projects what the state will be at the end of

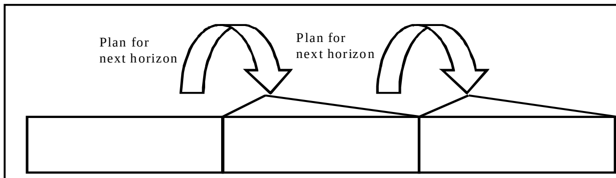

Figure 1 Traditional Batch "Plan then Execute" Cycle

the execution of the current plan. The planner is invoked with a new set of goals and this state as the initial state (for example the Remote Agent Experiment operated in this fashion (Pell et al, 1997)) .

This approach has a number of drawbacks. In this batch oriented mode, typically planning is considered an off-line process which requires considerable computational effort and there is a significant delay from the time the planner is invoked to the time that the planner produces a new plan.[1] If a negative event occurs (e.g., a plan failure), the response time until a new plan may be significant. During this period the system being controlled must be operated appropriately without planner guidance. If a positive event occurs (e.g., a fortuitous opportunity, such as activities finishing early), again the response time may be significant. If the opportunity is short lived, the system must be able to take advantage of such opportunities without a new plan (because of the delay in generating a new plan). Finally, because the planning process may need to be initiated significantly before the end of the current planning horizon, it may be difficult to project what the state will be when the current plan execution is complete. If the projection is wrong the plan may have difficulty.

To achieve a higher level of responsiveness in a *dynamic planning* situation, we utilize a *continuous planning* approach and have implemented a system called CASPER (for Continuous Activity Scheduling Planning Execution and Replanning). Rather than considering planning a batch process in which a planner is presented with goals and an initial state, the planner has a current goal set, a plan, a current state, and a model of the expected future state. At any time an incremental update to the goals, current state, or planning horizon (at much smaller time increments than batch planning)[2] may update the current state of the plan and thereby invoke the planner process. This update may be an unexpected event or simply time progressing forward. The planner is then responsible for maintaining a consistent, satisficing plan with the most current information. This current plan and projection is the planner's estimation as to what it expects to happen in the world if things go as expected. However, since things

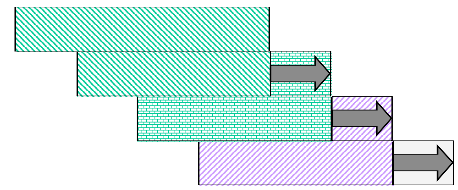

Figure2 Continuous Planning Incremental Extension

rarely go exactly as expected, the planner stands ready to continually modify the plan. From the point of view of the planner, in each cycle the following occurs:

- changes to the goals and the initial state first posted to the plan,
- effects of these changes are propagated through the current plan projections (includes conflict identification)
- plan repair algorithms[3] are invoked to remove conflicts and make the plan appropriate for the current state and goals.

This approach is shown in below in Figure 2. At each step, the plan is created by using iterative repair with:

- the portion of the old plan for the current planning horizon;
- the updated goals and state; and
- the new (extended)planning horizon.

Even though our intent is to make the planning process very responsive (on the order of seconds), there still remains a synchronization process between planning and execution. Specifically, there are several issues in integrating planning with real time execution - below we list these issues and how they are addressed in our approach.

- When to replan? Our approach replans when the current plan projection predicts a problem with the current plan (i.e. when the plan combined with the current state is infeasible).
- What to do (execute) during planning time? If feedback from the world combined with the current plan indicates that the current plan has a flaw (e.g., the plan will not execute or does not achieve goals), what gets executed during the time that the planner is replanning? *Our approach attempts to minimize the amount of time for replanning to minimize the chance that a conflict appears in the portion of the old plan that gets executed.*
- How much time should the planner be given to replan? The longer the planner is given the more likely it will be able to resolve all of the problems. But the longer the replan time, the greater the problem of "What to execute in the meantime?" from above. *Our approach attempts to minimize the*

[1] As a data point, the planner for the Remote Agent Experiment (RAX) flying on-board the New Millennium Deep Space One mission (Muscettola et al 1997) takes approximately 4 hours to produce a 3 day operations plan. RAX is running on a 25 MHz RAD 6000 flight processor and uses roughly 25% of the CPU processing power. While this is a significant improvement over waiting for ground intervention, making the planning process even more responsive (e.g., on a time scale of seconds or tens of seconds) to changes in the operations context, would increase the overall time for which the spacecraft has a consistent plan. As long as a consistent plan exists, the spacecraft can keep busy working on the requested goals and hence may be able to achieve more science goals.

[2] For the spacecraft control domain we are envisaging an update rate on the order of 10s of seconds real time.

[3] I n this paper we do not focus on the state/resource representation or the repair methods, for details see (Rabideau et al. 1999).

amount of time given to replan, but we believe this criteria can only be determined in a domain-specific fashion[4].

- How to ensure the planner does not change activities that are already in execution? *Our approach uses a commitment mechanism to represent activities that would not be changeable by the time that the planner would complete its current cycle of reasoning. When an activity overlaps with this window (i.e. the activity is scheduled to begin very soon) it is committed. This means that the planner is forbidden from altering any aspect of this activity (such as by moving the activity or altering the activity parameters). Thus far we have focused on time-based commitment strategies (e.g., commit any activities scheduled to begin in the next T time units), however, our architecture supports more complex commitment strategies (such as it being dependent on the class of activity and allowing parameter changes later than activity moves, etc.).*

In addition to increasing the responsiveness of planning, the continuous planning approach has additional benefits:

- The planner can be more responsive to unexpected (i.e., unmodeled) changes in the environment that would manifest themselves as updates on the execution status of activities as well as monitored state and resource values.
- The planner can reduce reliance on predictive models (e.g., inevitable modeling errors), since it will be updating its plans continually.
- Fault protection and execution layers need to worry about controlling the spacecraft over a shorter time horizon (as the planner will replan within a shorter time span).
- Because of the hierarchical reasoning taking place in the architecture there is no hard distinction between planning and execution – rather more deliberative (planner) functions reside in the longer-term reasoning horizons and the more reactive (execution) functions reside in the short-term reasoning horizons. Thus, there is no planner to executive translation process.

In conjunction with this incremental, continuous planning approach, we are also advocating a hierarchical approach to planning. In this approach, the long-term planning horizon is planned only at a very abstract level. Shorter and shorter planning horizons are planned in greater detail, until finally at the most specific level the planner plans only a short time in advance (just in time planning). This paradigm is illustrated in Figure 3. Within each of these layers, the planner is operating continuously in the mode described above. However, the length of the planning horizon, and the frequency with which the plan is updated varies. In the longer-term more abstract levels, the planning horizon is longer and the abstract plan is updated less frequently. In the more detailed short-term level, the plans are updated more frequently.

The idea behind this hierarchical approach is that only very abstract projections can be made over the long-term and that detailed projections can only be made in the short-term because prediction is difficult due to limited computational resources and timely response requirements. Hence there is little utility in constructing a detailed plan far into the future – chances are it will end up being re-planned anyway. At one extreme the short-term plan may not be "planned" at all and may be a set of reactions to the current state in the context of the near-term plan. This approach is implemented in the control loop described above by making high-level goals active regardless of their temporal placement, but medium and low-level goals are only active if they occur in the near future. Likewise, conflicts are only regarded as important if they are high-level conflicts or if they occur in the near future. As the time of a conflict or goal approaches, it will eventually become active and the elaboration/planning process will then be applied to resolve the problem.

An Architecture for Integrated Planning and Execution

Our approach to integration of planning and execution relies on three separate classes of processes.

- **<u>The Planner Process(es)</u>** - this process represents the planner, and is invoked to update the model of the plan execution, to refine the plan, or when new goals are requested.
- **<u>The Execution Process(es)</u>** - this process is responsible for committing activities and issuing actual commands corresponding to planned activities.
- **<u>The State Determination Process(es)</u>** - this process is responsible for monitoring and estimating states and resource values and providing accurate and timely state information.
- **<u>The Synchronization Process</u>** - this process enforces synchronization between the execution, planner, and state determination processes. This includes receiving new goals, determining appropriate timeslices for planning and locking the plan database to ensure non-interference between state updates and the planner.

We describe planning, execution, and state determination as sets of processes because often these logical tasks will

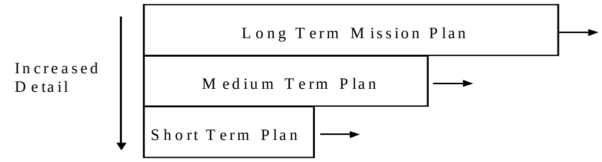

Figure 3 Hierarchical Planning Horizons

[4] However, an interesting area of research is to automatically determine this via empirical feedback and domain analysis.

be handled by multiple processes. For example, spacecraft attitude control execution might be handled by one process, data management by another, etc. However, for the purposes of this paper (e.g., integration of planning and execution), the only relevant issue is that our synchronization strategy can be applied to a multiple process scheme for planning, state determination, etc.

The overall architecture for the continuous planning approach is shown in Figure 4. We now describe how each of the four basic components operates.

The planner process maintains a current plan that is used for planning (e.g. hypothesizing different courses of action). It responds to requests to replan initiated by the execution processes, activity commitments vfrom the execution module, state (and resource) updates from state estimation, and new goals (from external to the system). All of these requests are moderated by the synchronization process that queues the requests and ensures that one request is complete before another is initiated. The planners copy of the current plan is also where projection takes place and hence it is here that future conflicts are detected. However, as we will see below, requests to fix conflicts occur by a more circuitous route.

The execution process is the portion of the system concerned with a notion of "now". The execution module maintains a copy of the plan that is incrementally updated whenever the planner completes a request (e.g., a goal change, state change, or activity change). This local copy includes conflict information. The execution module has three general responsibilities:

1. to commit activities in accordance with the commitment policy as they approach their execution time;
2. to actually initiate the execution of commands (e.g., processes) at the associated activity start times
3. to request re-planning when conflicts exist in the current plan

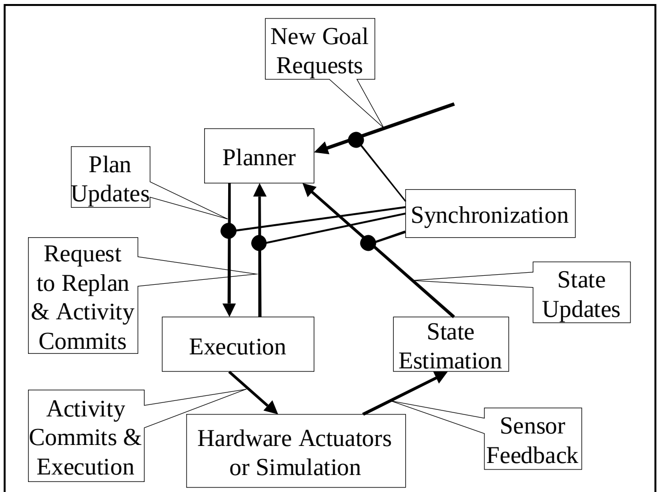

Figure 4: CASPER Architecture

The execution module performs 1 & 2 by tracking the current time and indexing into relevant activities to commit and execute them. The execution module also tracks conflict information as computed by the projection of the planner and submits a request for replanning to the synchronization module when a conflict exists.[5]

The state estimation module is responsible for tracking sensor data and summarizing that information into state and resource updates. These updates are made to the synchronization module that passes them on to the planners plan database when coordination constraints allow.

The synchronization module ensures that the planner module(s) are correctly locked while processing. At any one time the planner can only be performing one of its four responsibilities: (re)planning, updating its goals, incorporating a state update, updating the execution module's plan for execution, or updating commitment status (otherwise we run the risk of race conditions causing undesirable results). The synchronization module serializes these requests by maintaining a FIFO task queue for the planner and forwarding the next task only when the previous task has finished.

The execution module also has a potential synchronization issue. The planner must not be allowed to modify activities (through replanning) if those activities might already have been passed on to execution. We enforce this non-interference by "commit"-ing all activities overlapping a temporal window extending from now to some short period of time in the future (typically on the order of several seconds). We ensure that the planner is called in a way that each replan request will always return within this time bound and we enforce that the planner never modifies a committed activity. This ensures that the planner will not complete a replan with an activity modified that is already in the past. Additionally, we use the synchronization process to ensure that the Execution module does not commit activities while the planner is replanning. This prevents the planner from modifying activities that have been committed subsequent to the planner call (but still in the future).

ST4 Spacecraft Landed Operations Scenario Validation

Space Technology 4 / Champollion (ST4) is a mission concept for outer solar system exploration. In late 2005, following a two-and-a-half-year journey, ST4 will match

[5] In our implementation replanning is initiated by the execution module because this allows for the notion of urgency information (e.g. closeness of the conflict to current execution) to be incorporated in the decision to replan. If we did not wish to incorporate this information, the planner module could make this request directly to the synchronization module.

orbits, or rendezvous, with Comet Tempel 1, as the comet is moving away from the Sun. The spacecraft will spend several months orbiting the comet nucleus, making highly accurate maps of its surface and making some preliminary compositional measurements of the gas in the coma. The data returned from ST4 will be used to determine the mass, shape, and density of the comet's nucleus and to make some early estimates about its composition.

After studying the nucleus from orbit, the spacecraft will send a small vehicle (a lander) to the surface. The lander will use a one-meter long drill to collect samples and then feed them to a gas chromatograph/mass spectrometer onboard the lander. This instrument will analyze the composition of the nucleus collected from various depths below the surface. The lander will also carry cameras to photograph the comet surface. Additional instruments planned onboard the lander to determine the chemical makeup of the cometary ices and dust will include an infrared/spectrometer microscope and a gamma-ray spectrometer. After several days on the surface, the lander will bring a sample back to the orbiter for return to Earth.

In order to test and evaluate our integrated planning and execution approach, we have constructed a number of test cases within the ST4 landed operations scenario. We have also constructed an ST4 simulation, which accepts relatively high-level commands such as: MOVE-DRILL, START-DRILL, STOP-DRILL, TAKE-PICTURE, TURN-ON <device>, etc. The simulation covers operations of hardware devices. In this test scenario the planner has models of 11 state and resource timelines, including drill location, battery power, data buffer, and camera state. The model also includes 19 activities such as uplink data, move drill, compress data, take picture, and perform oven experiment.

The continuous planner scenario has focused on the comet lander portion of the ST4 mission. It comprises a period of approximately 80 hours of lander operations on the comet surface. It is intended to represent a class of test cases against which to evaluate the performance of various command and control strategies for this portion of the mission.

The nominal mission scenario consists of three major classes of activities: drilling and material transport, instrument activity including imaging and in-situ materials experiments, and data uplink. Of these, drilling is the most complex and unpredictable.

The mission plan calls for three separate drilling activities. Each drilling activity drills a separate hole and acquires samples at three different depths during the process: a surface sample, a 20 cm. deep sample, and a one-meter deep sample. Acquiring a sample involves five separate "mining" operations after the hole has been drilled to the desired depth. Each mining operation removes 1 cm. of material. Drilling rate and power are unknown a priori, but there are reasonable worst-case estimates available. Drilling can fail altogether for a variety of reasons.

One of the three drilling operations is used to acquire material for sample-return. The other two are used to supply material to in-situ science experiments onboard the lander. These experiments involve depositing the samples in an oven, and taking data while the sample is heated. Between baking operations the oven must cool, but there are two ovens, allowing experiments to be interleaved unless one of the ovens fails.

In order to validate the effectiveness of our continuous planning approach we have performed a number of empirical tests to measure CASPER performance in terms of:

1. responsiveness - the ability to deal with execution feedback in a timely fashion;
2. robustness - the ability to produce executable plans despite run-time variations in state, resource, and activity durations; and
3. plan effectiveness - a measure of the overall goodness of executed activities (with respect to achieving plan goals).

We assessed these performance metrics using a stochastic version of the ST4 simulation described above. This simulation had a number of random variables, which are described below.

- Compression - we model the compression for science data as a normal random variable with a mean of 0.9 and a standard deviation of 0.25*0.9. This has the effect of forcing the planner to respond to buffer over-runs (as described above) and buffer under-runs (to optimize the plan).
- Drilling Time - we model the amount of time to drill in minutes as a random variable with mean of 30 and standard deviation of 3.
- Drilling power - we model the actual power consumption from drilling in watts as a normal random variable with mean 40 and standard deviation 4.
- Oven Failure - we model oven failure occurrence as Poisson distributed with each oven having a 50% chance of failure over the entire mission horizon.
- Data Transmission Rate: we model the time to transmit data in kilobits per second as a normal random variable with a mean of 100 and a standard deviation of 10. This is intended to model the variability in communications to the orbiter.
- Oven Warming and Cooling Times: we model the amount of time to heat up the sample and for the oven to cool down in minutes as random variables with means of 30 and 120, and standard deviations of 3 and 12, respectively. This is intended to model the unknown thermal properties of the samples.

In our tests we compare the CASPER continuous planning repair approach to two alternative approaches:

1. Batch planning with no feedback - in this approach an operations plan is generated from the initial state and this plan is executed. No feedback from execution is used.
2. Batch replanning on failure - in this approach an operations plan is generated from scratch. When an activity fails, the execution system halts execution

and replans from scratch (rather than modifying the existing plan as in the CASPER approach). No activities are executed while the planner is replanning.

In all cases, we compare the approaches using models with best guess nominal estimates for times and resource usage, as well as pessimistic 1-sigma estimates.

In order to assess the responsiveness of the system, we measured the average amount of time from the receipt of an update that required replanning to the time when a conflict free plan is available (see Tables 1, 2 and 3: Time to Correct Plan). In order to assess the robustness of the system, we track the number of times when an invalid activity is commanded (see Tables 1, 2 and 3: Number of Invalid Commands). In order to assess the plan effectiveness, we measure the science return of executed activities (as measured by number of samples drilled and analyzed in situ where the data was successfully transmitted to the orbiter) 24 science goals are originally submitted to the system, and we report the number completed successfully. (See Tables 1, 2 and 3: Number of Achieved Science Goals).

In our setup, CASPER was running on a Sun Sparcstation Ultra 60 with a 359 MHz process with 1.1 GB Memory. During each run, the simulator updates the plan an average of 18,000 times. (Most of these are battery power level updates.) On average, only 86 updates result in conflicts that should be handled by the planner/scheduler.

We observe that CASPER outperforms batch planning and batch replanning in the ST4 domain in terms of spacecraft commanding and achieving science goals. Note that batch planning requires no time to correct an updated plan because it does not replan, and therefore is superior to CASPER in terms of the amount of time required to correct a plan. However, batch planning suffers considerably due to incomplete data transmissions and spoiled experiments where samples where placed into inappropriately configured or failed ovens. Batch replanning performs much better, but the replan time translates into missed opportunities to plan and schedule science goals. Also, more invalid commands are executed due to the time it takes to replan. CASPER does execute some invalid commands due to the fact that it takes some time to correct an invalid plan, but CASPER achieves far more science goals.

Overall Performance	# invalid commands	# achieved science goals	time to correct plan (in seconds)
CASPER	2.365	20.063	1.134
Batch planning	54.769	2.194	0
Batch replanning	17.977	6.722	20.125

Table 1 Overall Performance Comparison Averages

Best Guess Performance	# invalid commands	# achieved science goals	time to correct plan (in seconds)
CASPER	2.909	24.677	.978
Batch planning	61.341	2.699	0
Batch replanning	22.112	16.878	17.107

Table 2 Best Guess Performance Comparison

Pessimistic Performance	# invalid commands	# achieved science goals	time to correct plan (in seconds)
CASPER	1.821	15.448	1.291
Batch planning	48.197	1.690	0
Batch replanning	13.842	10.566	23.144

Table 3 Pessimistic Performance Comparison Averages

Discussion

While the current prototype has been tested on a range of cases in which state updates require replanning, we have focused on execution feedback that cause conflicts in the plan. In the case of the failed oven, buffer over-use, and activity completion time problem, the state update (when propagated through the plan) causes a conflict. There are other cases in which a state update enables a plan improvement. For example,

- battery power usage might be lower than expected enabling insertion of an additional sample activity content-dependent compression might perform better than expected allowing storage of additional experiment data; or
- drilling might be faster than expected again allowing for additional science activities.

In each of these cases, the planner needs to be aware of the potential for improvement in the current plan and be triggered to attempt to take advantage of the fortuitous situation. In related work (Rabideau et al. 2000), we have been developing plan optimization techniques for representing soft constraints (preferences) and improving plans with respect to these preferences (e.g., do more science). Our approach to optimization is an anytime, incremental approach, thus the timeslices for the planner can be used to attempt to improve the plan if there are no conflicts in the plan.

A second issue is that in the current prototype, the planner can only respond to unexpected changes on activity boundaries. This is a significant limitation when there are activities that have extremely long durations. This limitation is because the planner does not have a model detailed enough to predict the resultant state if activities are interrupted in mid-execution. It would be useful if the planner could incorporate a model that could represent interruptible activities and act appropriately. Currently such phenomenon must be modeled by breaking the activity into smaller activities.

While we have tested our prototype on a range of realistic scenarios, we would like to have a larger set of missions and concepts to test against. Because CASPER is currently being used for autonomous rover applications, we are in the process of adapting rover simulations for similar testing. Additionally we anticipate having access to several other spacecraft simulations. We intend to further test and validate our approach against these missions.

Another interesting area for future work is investigating more powerful commitment strategies. One could easily envisage problems in which different classes of activities would have different possibilities for interruption or might be terminatable with sufficient lead-time. Enabling the planner to represent these contexts and handle them appropriately would be desirable.

Related Work

The high-speed local search techniques used in our continuous planner prototype are an evolution of those developed for the DCAPS system (Chien et al. 1999) that has proven robust in actual applications. In terms of related work, iterative algorithms have been applied to a wide range of computer science problems such as traveling salesman (Lin & Kernighan 1973) as well as Artificial Intelligence Planning (Biefeld & Cooper 1991, Chien & DeJong 1994, Zweben et al. 1994, Hammond 1989, Sussman 1973). Iterative repair algorithms have also been used for a number of scheduling systems. The GERRY/GPSS system (Deale et al. 1994, Zweben et al. 1994) uses iterative repair with a global evaluation function and simulated annealing to schedule space shuttle ground processing activities. The Operations Mission Planner (OMP) (Biefeld & Cooper 1991) system used iterative repair in combination with a historical model of the scheduler actions (called chronologies) to avoid cycling and getting caught in local minima. Work by Johnston and Minton (Johnston & Minton 1994) shows how the min-conflicts heuristic can be used not only for scheduling but also for a wide range of constraint satisfaction problems.

The OPIS system (Smith 1994) can also be viewed as performing iterative repair. However, OPIS is more informed in the application of its repair methods in that it applies a set of analysis measures to classify the bottleneck before selecting a repair method. With iterative repair and local search techniques, we are exploring approaches complementary to backtracking refinement search approach used in the New Millennium Deep Space One Remote Agent Experiment Planner (ARC 1999).

Excalibur (Narayek, 1998) represents a general framework for using constraints to unify planning and scheduling constraints, uncertainty, and knowledge. This framework is consistent with the CASPER design, however in this paper we have focused on a lower-level. Specifically, we have focused on re-using the current plan using iterative repair and specific locking mechanisms to avoid race conditions.

Work on the PRODIGY system (Cox & Veloso 1998) has indicated how goals may be altered due to environmental changes/feedback. These changes would be modeled in our framework via task abstraction/retraction and decomposition for potentially failing activities. Other PRODIGY work (Veloso, Pollack, & Cox 1998) has focused on determining which elements of world state need to be monitored because they affect plan appropriateness. In our approach we have not encountered this bottleneck, our fast state projection techniques enable us to detect relevant changes by noting the introduction of conflicts into the plan.

Work on CPEF (Continuous Planning and Execution Framework) (Myers 1998) uses PRS, AP, and SIPE-2, also represents a similar framework to integrating planning and execution. CPEF and CASPER differ in a number of ways. First, CPEF attempts to cull out key aspects of the world to monitor (as is necessary in general open-world domains). They also suggest the use of iterative repair (they use the term conservative repairs). And their taxonomy of failure types is very similar to ours in terms of action failure and re-expansion of task networks (re-decomposition). However, in this paper we have focused on lower level issues in synchronization and timing.

Work in the O-Plan system has also addressed rapid replanning (Drabble et al. 1997). They describe an approach that generally invokes the planner with the current plan in a repair mode from the current state. In this way their approach and the CASPER one are very similar. However, we have focused on lower-level timing and synchronization issues necessary for execution and planning on a shorter timescale.

Work in the 3T system (Bonasso et al. 1997) has also examined issues of integrating planning and execution. Again, they present a framework consistent with our architecture but we have focused on lower-level timing issues.

Conclusions

This paper has described an approach to integrating planning and execution for spacecraft control and operations. This approach has the benefit of reducing the amount of time required for an onboard planning process to respond to changes in the environment or goals. In our approach, environmental changes or inaccurate models cause updates to the current state model and future projections. Additionally, the planner's current goal set may change. In either case, if these changes matter (e.g., the current plan no longer applies) they will cause conflicts in the current plan. These conflicts are attacked using fast, local search and iterative repair methods

Acknowledgements

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Portions of this work were supported by: the Autonomy Technology Program, managed by Dr. Richard Doyle and with Melvin Montemerlo as the headquarters program executive, NASA Code SM and by the Mission Data Systems Project, managed by Allan Sacks, NASA Code S, and by the JPL Telecommunications and Mission Operations Technology Program, Mission Services Element managed by Peter Shames. Comments and feedback from Bob Rasmussen, Kim Gostelow, Dan Dvorak, Erann Gat, Glenn Reeves, and Ed Gamble were very helpful in formulating and refining the ideas presented in this paper.

References

NASA Ames & JPL, Remote Agent Experiment Web Page, http://rax.arc.nasa.gov/, 1999.

E. Biefeld and L. Cooper, "Bottleneck Identification Using Process Chronologies," *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

R. P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, M. Slack, "Experiences with an architecture for intelligent, reactive agents. Journal of experimental and theoretical artificial intelligence 9(2).

S. Chien and G. DeJong, "Constructing Simplified Plans via Truth Criteria Approximation," *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Chicago, IL, June 1994, pp. 19-24.

S. Chien, G. Rabideau, J. Willis, and T. Mann, "Automating Planning and Scheduling of Shuttle Payload Operations," *Artificial Intelligence Journal*, 114 (1999) 239-255.

M. Cox & M. Veloso, "Goal Transformation in Continuous Pannning," in Proceedings of the AAAI Fall Symposium on Distributed Continual Planning, 1998.

M. Deale, M. Yvanovich, D. Schnitzius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, and B. Daun, "The Space Shuttle Ground Processing System," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

B. Drabble, J. Dalton, A. Tate, "Repairing Plans on the Fly," Working Notes of the First International Workshop on Planning and Scheduling for Space, Oxnard, CA 1997.

A. Fukunaga, G. Rabideau, S. Chien, D. Yan, "Towards an Application Framework for Automated Planning and Scheduling," *Proc. 1997 Int.l Symp. on Art. Int., Robotics and Automation for Space*, Tokyo, Japan, July 1997.

K. Hammond, "Case-based Planning: Viewing Planning as a Memory Task," Academic Press, San Diego, 1989.

M. Johnston and S. Minton, "Analyzing a Heuristic Strategy for Constraint Satisfaction and Scheduling," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

H. Kautz, B. Selman, "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search," *Proceedings AAAI96*.

S. Lin and B. Kernighan, "An Effective Heuristic for the Traveling Salesman Problem," *Operations Research Vol. 21*, 1973.

D. Mittman (mission operations and planning lead for Space Infra-red Telescope (SIRTF) Mission, personal communications, April 1997.

N. Muscettola, B. Smith, S. Chien , C. Fry , K. Rajan, S. Mohan, G. Rabideau , D. Yan, "On-board Planning for the New Millennium Deep Space One Spacecraft," *Proceedings of the 1997 IEEE Aerospace Conference*, Aspen, CO, February, 1997, v. 1, pp. 303-318.

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

A. Nareyek, "A Planning Model for Agents in Dynamic and Unicertain Real-Time Environments," in Integrating Planning, Scheduling, and Execution in Dynamic and Uncertain Environments, AIPS98 Workshop, AAAI Technical Report WS-98092.

B. Pell, D. Bernard, S. Chien, E. Gat, N. Muscettola, P. Nayak, M. Wagner, and B. Williams, " An Autonomous Spacecraft Agent Prototype," *Autonomous Robots,* March 1998.

G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," Int Symp on Artificial Intelligence Robotics and Aut. in Space (ISAIRAS), Noordwijk, The Netherlands, June 1999.

G. Rabideau, B. Engelhardt, S. Chien, "Using Generic Preferences to Incrementally Improve Plan Quality," in Proc. 5th Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-2000), Breckenridge, CO. April,2000.

R. Ridenoure, New Millennium Mission Operations Study (and Personal Communication to Guy Man), June 1995.

R. Simmons, "Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems," Technical Report, MIT Artificial Intelligence Laboratory, 1988.

S. Smith, "OPIS: An Architecture and Methodology for Reactive Scheduling," in *Intelligent Scheduling*, Morgan Kaufman, 1994.

G. Sussman, "A Computational Model of Skill Acquisition," Technical Report, MIT Artificial Intelligence Laboratory, 1973.

M. Veloso, M. Pollack, M. Cox, "Rationale-based monitoring for planning in dynamic environments," Proceedings Artificial Intelligence Planning Systems Conference, Pittsburgh, PA, 1998.

M. Zweben, B. Daun, E. Davis, and M. Deale, "Scheduling and Rescheduling with Iterative Repair," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.